Spektrumbecslés nem-egyenletesen mintavételezett adatokból DSP-n

TDK dolgozat

Szilvási Sándor

Konzulens:

Molnár Károly

2007

Tartalomjegyzék

1. fejezet

Bevezetés

Jelen dolgozat a méréstechnika és az elosztott jelfeldolgozás egy ehhez kapcsolódó érdekes problémájával foglalkozik. A dolgozat első fele a spektrumszámítás lehetőségeit vizsgálja meg abban a speciális esetben, amikor a mért jel egyenletes mintavételezése nem biztosítható, de a méréshez tartozó időpontok pontosan megállapíthatók.

Ez napjainkban azért bír jelentőséggel, mert például a IEEE 1451 szabvány [1] elterjedésével egyre jobban előtérbe kerülnek a hálózatba kapcsolható szenzorok. Az IEEE 1451 szabványcsomag az intelligens jelátalakító interfész (Smart Transducer Interface) általánosan elfogadott, nyílt szabványainak egy gyűjteménye szenzorokhoz és beavatkozókhoz. Az ilyen szenzorokból készíthetők olyan hálózatra kapcsolható egységek (Network Capable Application Processor, NCAP), melyek egyből csatlakoztathatók helyi IEEE 802.3 (Ethernet) hálózatra, vagy akár közvetlenül az Internetre. Ezek a hálózatok sok esetben már ki vannak építve, így kézenfekvő ötlet ezek használata szenzorhálózat kialakításakor. Az Ethernet hálózat nagy hátránya ugyanakkor, hogy nem lehet a csomagküldési időt garantálni.

Léteznek módszerek arra, hogy az Ethernet típusú hálózaton az egységek között időszinkronizácót valósítsunk meg, és azok így pontosan regisztrálni tudják a mintavétel időpontját, például az IEEE 1588 szabvány [2] alapján. A mintavételezés egyenletességét ugyanakkor nem mindig lehet biztosítani, ezért célszerű megvizsgálni, hogy a jelfeldolgozó rendszerekben ezt a problémát hogyan lehet kezelni.

A dolgozat bemutatja, hogy a nem-egyenletes mintavételezés problémája milyen alkalmazási területeken jelentkezik. Rendszerezi a mintavételezés egyenletlenségével kapcsolatos problémákat és összefoglalja, hogy az egyes hibatípusoknál, az adott alkalmazásnak megfelelően milyen módszert érdemes használni. Ezután egy szabadalomban megjelent új eljárást ismertet és hasonlít össze számítógépes szimulációk segítségével egy, a gyakorlatban gyakran al-

kalmazott módszerrel.

A dolgozat második fele ennek az új eljárásnak egy lehetséges implementációját mutatja be, ismertetve az ehhez szükséges hardver és szoftver rendszerterveket. Végül a megvalósított demonstrációs rendszerrel készített méréseket ismerteti és veti össze a korábbi szimulációs eredményekkel.

A megvalósítás elkészítése során két fő célkitűzésem volt. Egyrészt a rendszert úgy alakítottam ki, hogy alkalmas legyen Ethernet típusú hálózati kommunikációra. Ezáltal lehetőség van elosztott jelfeldolgozási problémák vizsgálatára, valamint más típusú hálózatok és az azokon fellépő kommunikációs hibák szimulálására is. Másrészt nagy hangsúlyt fektettem az elkészített szoftver modularitására, így biztosítva lehetőséget az egyes szoftverkomponensek – legyen az a hálózati kommunikációt, vagy a jelfeldolgozást végző modul – egymástól és a rendszer többi elemétől függetlenül történő módosítására vagy cseréjére.

2. fejezet

Spektrumbecslés nem-egyenletesen mintavételezett adatokból

A jelfeldolgozó alkalmazásokban sokszor van szükség a jel frekvenciatartománybeli jellemzésére. A spektrumot a hagyományos, elterjed és hatékony módszerek általában csak egyenletesen vett minták alapján tudják becsülni.

A spektrum származtatásának leggyakoribb módja, hogy a jelet Fourier-transzformáljuk:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}\mathrm{d}t \tag{2.1}$$

A gyakorlatban általában a jel mintáinak egy sorozatát gyűjtjük össze, majd azt diszkrét Fourier-transzformáljuk (Discrete Fourier Transform, DFT). A DFT számításának képlete a Fourier-transzformáció egyenletéből kiindulva az integrált téglányösszeggel közelítve könnyen meghatározható. A DFT a (2.1) számítását véges számú minta alapján végzi, tehát az integrált csak véges időintervallumon becsli. Feltéve, hogy az összegyűjtött minták száma N, a minták közötti távolság pedig mindig Δt, a (2.1) integrál az alábbi összeggel közelíthető

$$X(f_k) = \sum_{i=0}^{N-1} x(i\Delta t)e^{-j2\pi f_k i\Delta t}\Delta t \tag{2.2}$$

ahonnan Δt egységnyi választásával:

$$X_k = \sum_{i=0}^{N-1} x(i)e^{-j2\pi\frac{ki}{N}} \qquad k = 0, 1, ..., N-1 \tag{2.3}$$

A DFT kiszámítására hatékony algoritmus létezik, ez a jól ismert Fast Fourier Transform (FFT). A DFT-t a (2.3) egyenlet alapján közvetlenül $O(N^2)$, FFT-vel pedig $O(N \log N)$ lépésben lehet kiszámítani. A DFT és így az FFT algoritmus egyenletesen mintavételezett jelet, azaz állandó Δt értéket feltételez, így *közvetlenül* egyik módszer sem alkalmazható nem-egyenletesen mintavételezett jelek feldolgozására.

A szabálytalan időközönként mért minták spektrumanalízisére számos területen szükség van. Az egyik ilyen terület például a csillagászat, ahol a mérések ritkán és nagyon egyenletlen időközönként kerülnek rögzítésre. Ekkor a spektrumszámítás tipikusan offline módszerekkel történik, hiszen egy új minta viszonylag ritkán kerül rögzítésre, ami lehetőséget biztosít nagy erőforrásigényű számítások elvégzésére. Beágyazott vagy szenzorhálózatos környezetben ezek a módszerek nem alkalmazhatóak, mert a rendelkezésre álló erőforrások korlátozottak.

Jelen dolgozatban olyan módszerek kerülnek bemutatásra, melyek alkalmasak lehetnek szerényebb erőforrásokkal rendelkező rendszerekben történő on-line felhasználásra. Ezek közül az alábbi három módszert tekintjük át részletesebben:

- *Újramintavételezés és FFT.* A mérnöki gyakorlatban talán leggyakrabban alkalmazott módszer. Elsősorban az ún. hiányzó adat (missing data) jelensége miatt fellépő egyenletlen mintavételezés esetén ad jó eredményt.

- *Lomb periodogram.* A jel valószínűség-sűrűségfüggvényét becsli. A minták nagyon egyenletlen időbeli eloszlása esetén használható jól. Létezik gyors kiértékelést lehetővé tevő implementációja.

- *CDFT algoritmus.* Egy Lee A. Barford által jegyzett szabadalomban [3] bemutatott új módszer, időbélyeggel ellátott, hálózaton lekérdezett adatok alapján történő spektrumbecslésre.

A következő alfejezetekben ezek részletes bemutatása következik.

2.1. Újramintavételezés és FFT

A legtöbb jelfeldolgozó alkalmazásban az x_t jel egyenletesen mintavételezett

$$x_i \equiv x(t_0 + i\Delta t) \qquad i = 0, ..., N \tag{2.4}$$

értékei alapján számítjuk a spektrális összetevőket. Itt t_0 a mérés kezdeti időpontja, N a minták száma és Δt a mintavételi periódus. Vannak esetek,

amikor a minták (2.4) szerinti egyenletességét nem tudjuk biztosítani. Ennek gyakori esete, hogy egy-egy minta nem áll rendelkezésre, kimarad. Ez az ún. *hiányzó minta (missing data)* probléma. Ez gyakorlatban például a hálózati kommunikáció hibájából adódóan könnyen bekövetkezhet. Kézenfekvő megoldás a hiányzó minták pótlása, majd az így kapott (2.4)-nek megfelelő, egyenlő távolságra elhelyezkedő minták alapján az FFT számítása. A hiányzó mintákat az őket körülvevő mért értékek alapján interpolációval becsülhetjük meg. Az interpolálás több modell alapján történhet:

- *Nulladrendű tartóval (Zero Order Hold, ZOH).* A hiányzó minta értékét a legközelebbi mintevételi pont értékével becsüljük.

$$\min_j \|t - t_j\| = \|t - t_i\| \quad \Rightarrow \quad x(t) = x(t_i) \tag{2.5}$$

  Itt j a rendelkezésre álló minták indexeit, t pedig a hiányzó adathoz tartozó időpontot jelöli.

- *Elsőrendű tartóval (First Order Hold, FOH).* A hiányzó minta értékét a két szomszédos mintára fektetett egyenesen vesszük fel.

$$t_i \leq t < t_{i+1} \Rightarrow x(t) = x_i + \alpha(t - t_i) \qquad \alpha = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \tag{2.6}$$

  Itt t_i és t_{i+1} a két szomszédos pont, t pedig a pótlandó minta idejét jelöli.

- *Lokálisan polinommal.* A mintavételi pontokra olyan alacsony fokszámú polinomot illesztünk, ami több környező mintavételi ponton is áthalad. A hiányzó adat idejét a polinomba behelyettesítve kapjuk a becsült értékét.

Az interpoláció számítási igénye a felsorolás sorrendjében növekszik.

Amennyiben a nem-egyenletesség más formában jelentkezik, azaz sok egymás utáni minta hiányzik, a minták csoportokba koncentrálva jelennek meg, vagy általában véletlenszerűen jönnek, ez a módszer már nem ad kielégítő eredményt. Például ha hosszabb ideig nincsenek minták, akkor a spektrumban a hiányzó szakasznak megfelelő hullámhosszú torzító koponensek jelennek meg.

2.2. Lomb periodogram

A nem-egyenletes mintákon alapuló spektrumanalízis egy, a hagyományos megközelítésektől nagyon eltérő módja a Lomb nevéhez fűződő módszer [9].

A módszer részletesen [10] 13. fejezetében kerül bemutatásra. A módszer sajátossága, hogy sztochasztikus jelmodellt feltételez, tehát a spektrális sűrűségfüggvényt becsli. Ennek neve a periodogram.

Tegyük fel, hogy N minta áll rendelkezésre, $x_i \equiv x(t_i),\ i = 1, ..., N$. Először határozzuk meg a mért jel középértékét és varianciáját a megszokott képletekkel

$$\overline{x} \equiv \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma^2 \equiv \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2 \tag{2.7}$$

Ezután a normalizált Lomb periodogramot a

$$P_N(\omega) \equiv \frac{1}{2\sigma^2}\left(\frac{\left(\sum_j (x_j - \overline{x})\cos\omega(t_j - \tau)\right)^2}{\sum_j \cos^2\omega(t_j - \tau)} + \frac{\left(\sum_j (x_j - \overline{x})\sin\omega(t_j - \tau)\right)^2}{\sum_j \sin^2\omega(t_j - \tau)}\right) \tag{2.8}$$

képlet adja meg, ahol τ az alábbi összefüggéssel van definiálva

$$\tan(2\omega\tau) = \frac{\sum_j \sin 2\omega t_j}{\sum_j \cos 2\omega t_j} \tag{2.9}$$

A τ érték egyfajta ofszet, bevezetésével $P_N(\omega)$ teljesen invariánssá válik a t_i mintavételi időpontok konstans idővel való eltolására. τ (2.9) szerinti definíciója más következménnyel is jár. A (2.8) kifejezés ekkor az ω frekvencián lévő harmonikus összetevőjének legkisebb négyzetek szerinti (least squares, LS) becslője, a

$$x(t) = A\cos\omega t + B\sin\omega t \tag{2.10}$$

modell szerint.

Az eljárás az *intervallumon* alapuló számítások helyett kifejezetten a mintavételi *pontok* alapján dolgozik, ami nem-egyenletes mintavételezés esetén jelentősen csökkenti a hibát. Ez a magyarázata annak, hogy a Lomb módszer miért tud nem-egyenletesen mintavételezett adatokon a hagyományos FFT-n alapuló módszereknél jobb eredményt produkálni.

A normalizált periodogram kiértékelése nem tartozik a gyors algoritmusok közé. Tipikusan N mintához $2N$, vagy $4N$ ponton szeretnénk meghatározni a periodogramot. Ez (2.8) és (2.9) kifejezéseknek megfelelően nem csak néhány összeadást és szorzást jelent, hanem további négy trigonometrikus függvény hívását. Ennek megfelelően az algoritmus műveletigénye $O(N^2)$. Ha a kiértékelendő frekvenciák egyenletesen helyezkednek el, a trigonometrikus függvények ismétlődése kihasználható optimalizálásra. Ezzel azonban legfeljebb kb. négyszeresére tudjuk növelni a sebességet.

Létezik a Lomb periodogram számítására egy hatékony implementáció. Ez a gyorsított Lomb periodogram számítás, ami a (2.8) és (2.9) egyenleteket csak közelítőleg számítja, de azt tetszőleges pontossággal. Az algoritmus FFT-hez hasonló módszert használ, de nem közvetlenül a periodogram számítására, hanem annak részszámításaihoz. Ennek részletes ismertetését lásd [10]. Ezzel az Lomb periodogram számításának lépésszáma $O(N \log N)$ nagyságrendre csökkenthető.

2.3. CDFT algoritmus

A nem-egyenletesen mintavételezett adatok alapján történő spektrumbecslés egy lehetséges módját egy szabadalom [3] közli, melyet Lee A. Barford jegyez. A szabadalom egy olyan számítási eljárást ismertet, ahol a nem-egyenletes időközönkénti mintavételezést hálózatba kötött szenzorok végzik, az adatok összegyűjtését és a spektrumbecslést pedig egy különálló egység hajtja végre. Ez az egység szintén a hálózaton keresztül kommunikál a szenzorokkal. A továbbiakban erre a feldolgozó egységben végrehajtott az eljárásra *Continuous Discrete Fourier Transform (CDFT) algoritmus* néven hivatkozunk. A módszer a nem-egyenletesen mintavételezett, időbélyeggel ellátott adatokból közvetlenül számol spektrumbecslőt.

Jelölje $t_0, t_1, ..., t_{N-1}$ a mintavételi időpontokat, az ezekhez tartozó mért értékeket pedig rendre $x_0, x_1, ..., x_{N-1}$, ahol $x_i = x(t_i)$. Feltételezzük, hogy a mintavételhez tartozó időbélyeg pontos, továbbá nem foglalkozunk az esetleges számábrázolási problémákkal.

Ahhoz hogy a (2.1) egyenlet kiértékelhető legyen, feltételezzük, hogy a jel sávkorlátozott. A mintavételek között a jel különböző módon közelíthető.

- *Nulladrendű tartóval (Zero Order Hold, ZOH).* A jel értéke a két mintevételi időpont között nem változik, azaz

$$t_i \leq t < t_{i+1} \Rightarrow x(t) = x(t_i) \tag{2.11}$$

- *Elsőrendű tartóval (First Order Hold, FOH).* A jel folyamatos, értéke a két mintavételi időpont között lineárisan változik.

$$t_i \leq t < t_{i+1} \Rightarrow x(t) = x_i + \alpha(t - t_i) \qquad \alpha = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \tag{2.12}$$

- *Lokálisan polinommal.* A mintavételi pontokra olyan alacsony fokszámú polinomok illeszthetők, melyek több környező mintavételi ponton is áthaladnak.

Vegyük azt az esetet, ahol a jel véges tartójú. Ekkor a Fourier transzformált

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}\mathrm{d}t = \int_{t_0}^{t_N} x(t)e^{-j2\pi ft}\mathrm{d}t \tag{2.13}$$

Az integrálást intervallumokra felbontva ez a következő alakra hozható

$$X(f) = \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} x(t)e^{-j2\pi ft}\mathrm{d}t \tag{2.14}$$

Ha a mintavételi pontok között ZOH modellt alkalmazunk, és a következő módon közelítünk

$$X(f) \simeq \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} x_i e^{-j2\pi ft}\mathrm{d}t = -\frac{1}{j2\pi f} \sum_{i=0}^{N-1} x_i (e^{-j2\pi f t_{i+1}} - e^{-j2\pi f t_i}) \tag{2.15}$$

A fenti számítást általában $N/2$ frekvenciára számoljuk ki, ami nagyságrendileg $O(N^2)$ lépésben végezhető el. Ez mérsékelt N értékekre elfogadható.

Most vizsgáljuk meg ugyanezt FOH modell esetén. Ekkor $X(f)$ értékét az alábbi összefüggéssel közelítjük

$$X(f) \simeq \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} \Big(\frac{x_{i+1} - x_i}{t_{i+1} - t_i}(t - t_i) + x_i \Big) e^{-j2\pi ft}\mathrm{d}t \tag{2.16}$$

Vezessük be az alábbi rövidítéseket

$$a = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \qquad k = -j2\pi f \tag{2.17}$$

Ekkor (2.16) így írható fel

$$
\begin{aligned}
X(f) &\simeq \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} (x_i - at_i + at)\, e^{kt} \mathrm{d}t = \qquad (2.18)\\
&= \sum_{i=0}^{N-1} \left(\int_{t_i}^{t_{i+1}} (x_i - at_i)\, e^{kt} \mathrm{d}t + \int_{t_i}^{t_{i+1}} at\, e^{kt} \mathrm{d}t \right) = \\
&= \sum_{i=0}^{N-1} \left(\left[\frac{x_i - at_i}{k}\, e^{kt} \right]_{t_i}^{t_{i+1}} + \left[a \left(\frac{t}{k} - \frac{1}{k^2} \right) e^{kt} \right]_{t_i}^{t_{i+1}} \right) = \\
&= \sum_{i=0}^{N-1} \left(\frac{x_i - at_i}{k} \left(e^{kt_{i+1}} - e^{kt_i} \right) + a \left(\frac{t_{i+1}}{k} - \frac{1}{k^2} \right) e^{kt_{i+1}} - \right. \\
&\qquad \left. - a \left(\frac{t_i}{k} - \frac{1}{k^2} \right) e^{kt_i} \right) \\
&= \sum_{i=0}^{N-1} \left(\left(\frac{x_i - at_i}{k} + a \left(\frac{t_{i+1}}{k} - \frac{1}{k^2} \right) \right) e^{kt_{i+1}} - \right. \\
&\qquad \left. - \left(\frac{x_i - at_i}{k} + a \left(\frac{t_i}{k} - \frac{1}{k^2} \right) \right) e^{kt_i} \right) = \\
&= \sum_{i=0}^{N-1} \left(\left(\frac{x_i}{k} - \frac{a}{k^2} \right) \left(e^{kt_{i+1}} - e^{kt_i} \right) + \frac{a}{k} \left(t_{i+1} - t_i \right) e^{kt_{i+1}} \right)
\end{aligned}
$$

Ebbe a és k értékét (2.17) szerint behelyettesítve kapjuk

$$
\begin{aligned}
X(f) &\simeq -\frac{1}{j2\pi f} \sum_{i=0}^{N-1} \left(\left(-\frac{1}{j2\pi f} \frac{x_{i+1} - x_i}{t_{i+1} - t_i} + x_i \right) \left(e^{-j2\pi f t_{i+1}} - e^{-j2\pi f t_i} \right) + \right. \\
&+ \left. (x_{i+1} - x_i)\, e^{-j2\pi f t_{i+1}} \right) \qquad (2.19)
\end{aligned}
$$

Ezt összevetve a (2.15) kifejezéssel, nagyságrendileg nem lett bonyolultabb, mert x_{i+1}, x_i, t_{i+1} és t_i rendelkezésre áll. Meg kell jegyezni, hogy a $t_{i+1} - t_i$ számítása numerikus problémát okozhat, ha értéke nagy.

Az algoritmus sajátossága, hogy a becslő mintánként számítható, szemben a többi Fourier-transzformációt végző eljárással (pl. FFT), ahol a kiértékelés tipikusan blokkosan történik.

A spektrumbecslő előállítása során a korábbi ütemben kiszámított részeredmények jelentős része felhasználható, amivel az algoritmus gyorsítható.

Ilyen részeredmények például az exponenciális kifejezések értékei. További optimalizálásra van lehetőség, ha a kiértékelendő frekvenciák a (2.20) szerint ekvidisztáns helyezkednek el.

$$f_k = kf_1 \qquad k = 0, 1, ..., N \tag{2.20}$$

Ekkor az exponenciális tagok kitevőinek számítása nagyban egyszerűsödik.

Ezek az optimalizálási lehetőségek a CDFT algoritmus implementációjának leírásánál, az 5.3 alfejezetben kerülnek részletes bemutatásra.

3. fejezet

Szimulációk

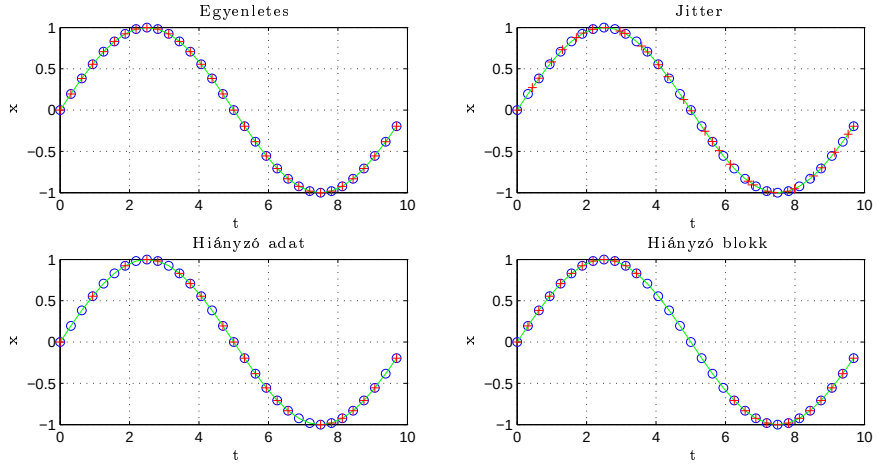
A nem egyenletesen rögzített mintákon alapuló spektrumbecslés DSP-n való kísérleti megvalósítására a korábbi fejezetben bemutatott módszerek közül a *CDFT algoritmust* választottam. Ennek oka, hogy a fenti módszerek közül ez a legújabb, alkalmazását tekintve ezzel kapcsolatban van a legkevesebb tapasztalat, ezért érdemes megvizsgálni és összevetni a többi metódussal. A kísérletezéseket először ideális környezetben, Matlab szimulációval végeztem annak érdekében, hogy a CDFT algoritmus sajátossága által behozott mellékhatásokat mennél egyértelműbben azonosíthassam. A szimulációk során a PC-n futó Matlab IEEE 754 szabványnak megfelelő, dupla pontosságú lebegőpontos számait használtam. Az így ábrázolt számokon végzett számítások a kísérletek során pontosságukat tekintve ideálisnak tekinthetők.

3.1. Az alap CDFT algoritmus vizsgálata Matlab szimulációval

A CDFT algoritmus vizsgálatát az hálózati jelfeldolgozásban előforduló nem-egyenletes mintavételezések kategorizálásával kezdtem. Ennek során három különböző típusú hibát különítettem el:

- *Jitter jellegű nem-egyenletesség.* A mintavételi időpontok távolságának átalaga állandó, de eloszlása egyenletes egy adott intervallumon.
- *Hiányzó adat (Missing data).* Az egyébként egyenletesen mintavételezett jelből elszórtan, bizonyos mintavételi pontok hiányoznak.
- *Blokkos kimaradás.* Az egyébként egyenletesen mintavételezett jelből egy több mintából álló, összefüggő tartomány hiányzik.

A szimulációk során kétféle vizsgálójelet alkalmaztam. Egyik esetben egyetlen szinuszjelet használtam és ennek detektálhatóságát vizsgáltam, míg a másik esetben multiszinusz segítségével vizsgáltam a spektrumbecslőt nagyobb terjedelmében. Ezeket a jeleket egyenletesen, valamint a fentebb felsorolt nem-egyenletes hibákkal mintavételeztem, azaz állítottam elő Matlabban. Ezt egy szinusz egyetlen periodusán mutatja be a 3.1. ábra.



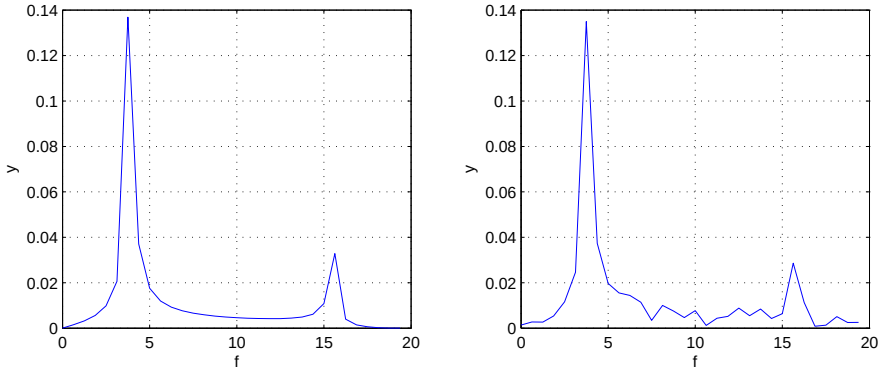
o – egyenletes mintavételi pontok
+ – szimulált nem-egyenletes mintavételi pontok

3.1. ábra. A különböző nem-egyenletes mintavételezések szimulációja

A CDFT algoritmus azon két változatához készítettem szimulációt, ahol a jel két mintavételi pont közötti értékét *nulladrendű (ZOH)* és *elsőrendű tartóval (FOH)* közelítjük. Összehasonlítás céljából az újramintavételezés és FFT eljárással is becsültem a spektrumot. Az újramintavételezést *nulladrendű* és *lineáris* interpolációval is elvégeztem.

Az első szimuláció során egy 4 Hz-es szinuszjelet vizsgáltam. A mintavételi frekvenciát 10 Hz-re állítottam. A szimuláció eredményét a 3.2. ábra tartalmazza. Az eredmények az algoritmus működőképességét igazolták, ezért nekiláttam a spektrum részletesebb vizsgálatának. A továbbiakban négy összetevőből álló multiszinusszal készített szimulációk eredményét mutatom be a különböző mintavételezés típus szerint csoportosítva.

Egyenletes mintavételezés. A szimulációt a jel egyenletes mintavételezéséből becsült spektrumok analízisével kezdtem, ezek alapján vizsgáltam meg az egyes módszerek alapvető sajátosságait. A szimuláció során $f_s = 25.6$ Hz mintavételi frekvenciát használtam és a négy szinuszjel frekvenciáját úgy

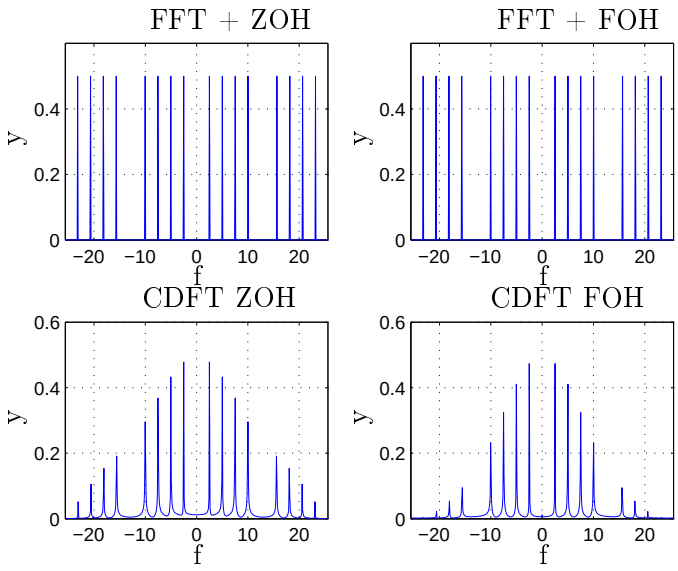


3.2. ábra. 4 Hz szinuszjel 10 Hz-es egyenletes (balról) és nem-egyenletes (jobbról) mintavételezéssel (Matlab szimuláció)

választottam meg, hogy a $[-f_s, f_s]$ tartományt egyenletesen töltsék ki. A különböző módszerek eredményezte spektrumbecslőket a 3.3. ábra mutatja.

A felső sorban található spektrumokat nézve azonnal szembetűnik, hogy a gerjesztőjel koherensen van mintavételezve. Ennek következtében az FFT-vel számolt felső két ábra pontosan visszaadja a periodikus komponenseket, nem jelentkezik sem a léckerítés (picket fence), sem a szivárgás (leakage) jelensége. Azért választottam ezt a speciális jelet, mert az FFT-vel így kapott spektrum könnyen összehasonlítható lesz más módszerek és a nem-egyenletesen mintavételezett jelek használatával kapott eredményekkel. A CDFT eredményeit ezekkel összehasonlítva azt tapasztaltam, hogy a spektrumbecslő a periodikus komponenseket torzítva ábrázolja, a nullfrekvenciás komponenstől a mintavételi frekvencia felé haladva csökkenő amplitúdóval. Ennek oka, hogy a spektrumot (2.15) szerint számítottuk. Ez az összefüggés annak felel meg, hogy a mintavétel után nulladrendű tartót alkalmazunk a következő minta beérkezéséig. Az így kapott folytonos idejű jelnek pedig (2.14) alapján diszkrét frekvenciaértékekre kiszámítjuk a Fourier-transzformáltját. Ennek megfelelően a periodikus komponensek nulladrendű közelítés esetén $\mathrm{sinc}(2\pi \mathrm{f}/\mathrm{f_s})$ értékkel szorzódnak, ld. 3.4. ábra. Elsőrendű közelítés esetén a mintavételi pontok közötti szakasz (2.16) szerint lineárisan közelítjük. Ekkor a spektrum $\mathrm{sinc}^2(2\pi \mathrm{f}/\mathrm{f_s})$ burkolójú csillapítást szenved, ld. 3.5. ábra. Ez azért előnyös, mert a magas frekvenciás torzító komponenseket jobban elnyomja.

Jitteres mintavételezés. A jitteresen $f_s = 25.6$ Hz átlagos mintavételi frekvenciával mért szinusz spektrumbecslői a 3.6. ábrán láthatók. A CDFT algoritmussal számított becslő minimális torzítást szenvedett az egyenletes esetben nyert eredményekhez képest, ld. 3.6 alsó sora. Az újramintavételezés és



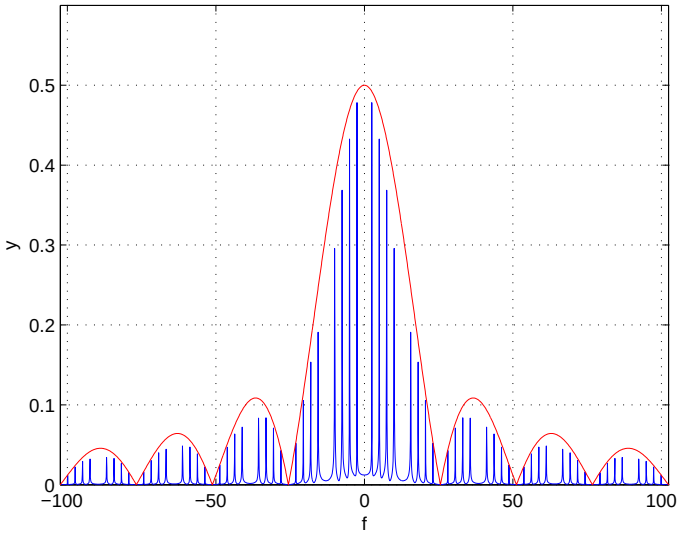
3.3. ábra. Egyenletesen mintavételezett multiszinusz spektrumbecslői

FFT módszer azonban már nem mutatja azt az egyenletessége, amit a 3.3. ábránál tapasztaltunk. Ennek oka, hogy az újramintavételezés itt is behozott nulladrendű esetben egy $\mathrm{sinc}(2\pi f/f_s)$, elsőrendű interpoláció esetében pedig egy $\mathrm{sinc}^2(2\pi f/f_s)$ jellegű burkolót.

Hiányzó adat. A nem-egyenletes mintavételezés ezen típusánál az egyenletesen mintavételezett adatok kb. 5%-a véletlenszerűen kimarad. Az így kapott spektrumok láthatók a 3.7. ábrán. Az eredmény a várakozásoknak megfelelő. Az FFT-vel számított eredményekben ismét megjelenik a spektrumban az interpolációk sinc és sinc^2 hatása. A CDFT algoritmus eredményei közel megegyeznek a korábbi szimulációknál kapottakkal.

Blokkos kimaradás. Az eredetileg egyenletesen mintavételezett adatokból véletlenszerűen meghatározott helyről minták összefüggő csoportja marad ki. A különböző módszerek eredménye a 3.8. ábrán láthatók. A 3.8. ábrán mindkét megközelítésnél esetében torzult a spektrum és megjelent egy minimális DC komponens.

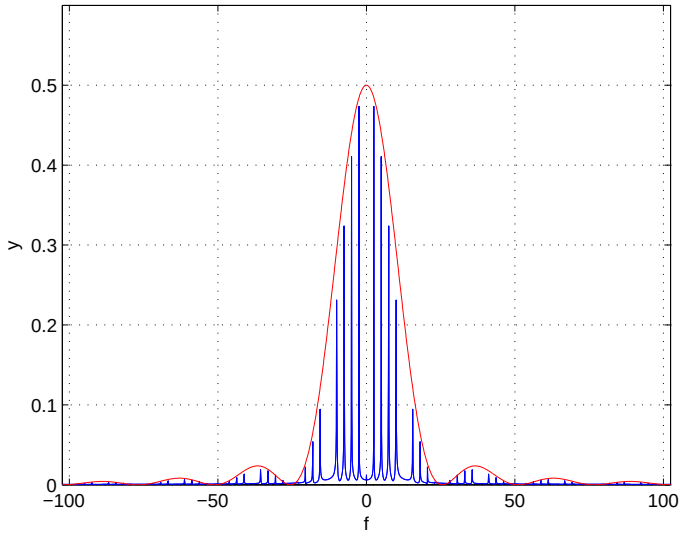
Összefoglalólag elmondható, hogy a két módszer nagyon hasonló eredményeket szolgáltat. Ennek oka, hogy mindkét eljárás hasonló megközelítést alkalmaz a hiányzó mintavételi pontok pótlására.



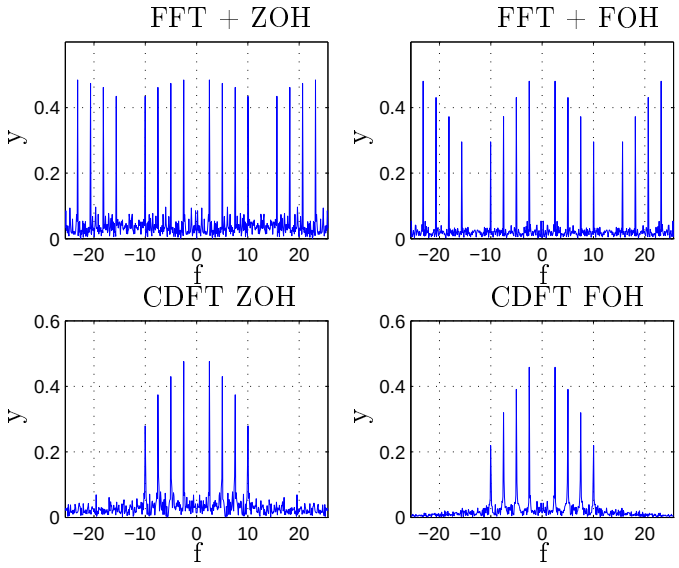
3.4. ábra. A CDFT ZOH algoritmus által számolt spektrum sinc burkolója $[-4f_s, 4f_s]$ tartományban

A CDFT algoritmus előnye, hogy a spektrumot közvetlenül számolja, szemben az FFT-s megoldással, ahol előzetes interpolációra van szükség. A közvetlen számítás árát a spektrum sinc torzulásának eltávolításakor fizetjük meg.

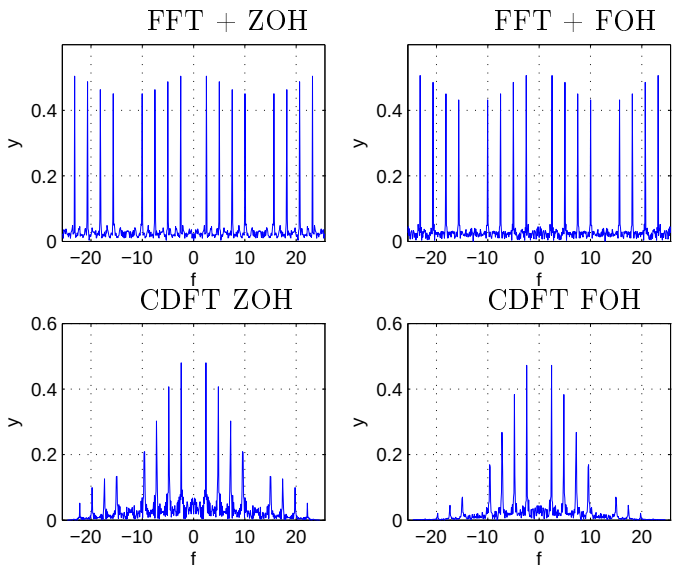
A két algoritmus futási idejét összehasonlítva az FFT-n alapuló eljárások gyorsabbnak bizonyultak. Ez egyrészt az FFT hatékonyságából, másrészt a CDFT algoritmus optimalizálás nélküli, képlet szerinti implementálásából fakad.



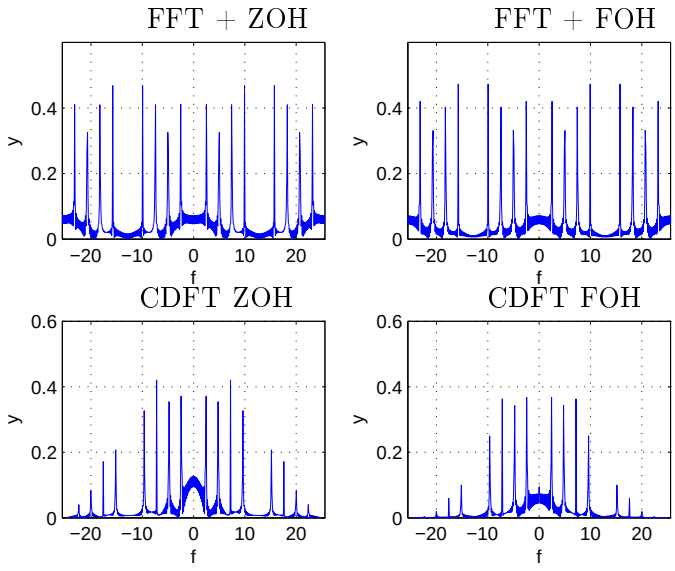
3.5. ábra. A CDFT FOH algoritmus által számolt spektrum sinc^2 burkolója $[-4f_s, 4f_s]$ tartományban



3.6. ábra. Jitteresen mintavételezett multiszinusz spektrumbecslői



3.7. ábra. Hiányosan mintavételezett multiszinusz spektrumbecslői



3.8. ábra. Csoportos kihagyással mintavételezett multiszinusz spektrumbecslői

4. fejezet

Próbarendszer megvalósításának eszközei

4.1. ADSP-BF537 EZ-KIT Lite fejlesztőkártya

A demonstrációs rendszer kialakítását az Analog Devices ADSP-BF537 jelfeldolgozó processzorára épülő EZ-KIT Lite [7] fejlesztőkártya segítségével végeztem. A fejlesztőkártya, a hozzá kapható kiegészítő modulok, a rajta található DSP architektúrája és a kártyához tartozó fejlesztőrendszer elsősorban gyors prototípus audio- és videoalkalmazások fejlesztésére alkalmas. A kártya perifériákban gazdag alapfelszereltsége azonban lehetővé teszi más területen való felhasználását. A döntésemet is ez indokolta: a tanszéken könnyen elérhető és segítségével gyorsan meg lehet ismerkedni az ADSP-BF537 processzor nyújtotta lehetőségekkel, továbbá nincs szükség külön tesztáramkörök fejlesztésére.

A kártya legfontosabb paraméterei, perifériái:

- ADSP-BF537 Blackfin processzor
- 64 Mbyte SDRAM memória
- 4 Mbyte Flash memória
- SMSC LAN83C185 10/100 PHY IEEE 802.3 szabványos Ethernet vezérlő RJ45 csatlakozóval
- AD1871 96 kHz sztereo DAC 3.5 mm jack csatlakozóval
- AD1854 96 kHz sztereo ADC 3.5 mm jack csatlakozóval
- USB alapú hibakereső (debug) interfész

- 6 általános célú LED és 4 nyomógomb

4.2. ADSP-BF537 jelfeldolgozó processzor

Az EZ-KIT Lite kártyán található ADSP-BF537 processzor [8] az Analog Devices cég Blackfin processzorcsaládjának tagja. A processzor egyszerre képes vezérlési és jelfeldolgozás típusú feladatok elvégzésére, így széles körben alkalmazható a beágyazott informatika különféle területein. Ezek közül kifejezetten ajánlott az alábbiakra:

- távoli eszközök monitorozása (remote monitoring)
- felügyeleti és biztonsági rendszerek
- ipari irányítási és automatizálási rendszerek

A fixpontos DSP utasításkészletét tekintve hibrid 16/32 bites RISC felépítésű, ahol az egyszerű utasításkészletet nagyfokú orthogonalitás jellemzi. Ezt a felépítést és az egy utasítással, több adaton dolgozó (Single Instruction Multiple Data, SIMD) technikát felhasználva a processzor lehetőséget biztosít a jelfeldolgozás során igen gyakran használt szorzás-összeadás (Multiply and ACcumulate, MAC) és számos képfeldolgozási művelet egy utasításciklus alatt történő, párhuzamos végrehajtására.

A Blackfin processzorok memóriaszervezését a módosított Harvard architektúra egy hierarchikus struktúrája jellemzi. Ennek megfelelően a processzor megkülönböztet kód- illetve adatmemóriát. A két memóriaterület külön buszon érhető el, ami a párhuzamos hozzáférés lehetőségét biztosítja. Ezt használják ki a SIMD utasítások, melyeket alkalmazva az időkritikus műveletek elvégzése felgyorsítható. A processzor által kezelt memóriák fizikai elhelyezkedését, és elérési sebességet nézve a belső (on-chip) memóriát egyetlen utasításciklus alatt azonnal, míg a külső (off-chip) memóriákat a külső busz interfész egységet használva, csak lassabban lehet elérni. Előbbi elsődleges memória három blokkból áll:

- 64 Kbyte SRAM, ami csak utasításokat tartalmazhat (és amiből 16 Kbyte opcionálisan cache-ként is használható)
- 2x32 Kbyte SRAM memóriabank, az adatok tárolására (opcionálisan cache-ként használható)
- 4 Kbyte SRAM

A külső memóriák típusa SDRAM, FLASH vagy SRAM lehet. Ezek a 32 bites címzéssel elérhető 4 Gbyte címtartományból összesen legfeljebb 516 Mbyte területet foglalhatnak el. A Blackfin család esetében az I/O regiszterek ugyanezen 4 GBytos címtartomány meghatározott részei, így elérésük is a memóriacímzésnél alkalmazott 32 bites címzéssel történik. A processzorban helyet kapott egy memóriakezelő egység (MMU) is, ami operációs rendszer használata esetén az egyes taszkok memóriaterületének védelmét felügyeli.

A processzorcsalád tagjai számos szabványos kommunikációs interfész megvalósításához nyújtanak hardveres támogatás. A BF537 esetében a beágyazott rendszereknél megszokottakon (pl. CAN 2.0B, UART) túl az IEEE 802.3 szabványnak megfelelő 10/100 Ethernet MAC protokollt is implementálták.

4.3. VisualDSP++ fejlesztői környezet

A fejlesztőrendszer nem csak az ADSP-BF537 processzort hordozó EZ-KIT Lite fejlesztőkártyát tartalmazza, hanem a fejlesztéshez elengedhetetlen szoftvereszközöket is, amiket az Analog Devices VisualDSP++ fejlesztői környezete fog egybe. A VisualDSP++ egy PC-n futó integrált fejlesztői környezet, ami az Analog Devices DSP alkalmazások írása és debuggolása céljából készült. Legfontosabb komponensei a C/C++ fordító, az assembler, a linker és a hibakeresést segítő modulok.

A VisualDSP++ fejlesztői környezet rendelkezik a beágyazott eszközök fejlesztésénél általában alkalmazott valamennyi szolgáltatással.

4.3.1. Projekt szervezés

A VisualDSP++ a forrásfájlokat struktúrált projektekbe szervezi, ami sok előnnyel jár:

- projektenként eltérő processzor-konfigurációkat, fordítási és linkelési paramétereket állíthatók be
- a külön fájlban megírt források könnyen öszerendelhetők, kezelhetők
- a fordításnál csak a módosult fájlok újrafordítása szükséges
- egyetlen gombnyomással újrafordíthatjuk (és letölthetjük) a programunkat a DSP-re

4.3.2. Fordító, assembler és linker

A VisualDSP++ támogatja a különböző nyelveken megírt fájlok együttes kezelését, egymásba ágyazását. A program megírható gépi (assembly), C, vagy C++ nyelven. Ez jól kihasználható a beágyazott szoftverek fejlesztésénél, ahol szokásos a programkódot felosztani időkritikus és nem-időkritikus részre. Itt a különböző nyelvek használatával lehetőség nyílik arra, hogy a nem-időkritikus szegmenseket magasabb szintű nyelven, jól olvasható, struktúrált módon írjuk meg, míg az időkritikus feladatokat könnyen beilleszthető assembly betétekkel implementáljuk. A fordítás és a gépi kód generálásának végterméke egyaránt egy tárgykódot tartalmazó, linkelhető fájl, amit a linker a linkelést leíró fájl alapján fűz össze. A linkelés eredményeként munkamemóriába tölthető, vagy EPROM-ba égethető fájl keletkezik.

A gyakran használt, fontosabb függvények futás-idejű könyvtárként (runtime library) állnak rendelkezésre [5], melyek a processzor assembly nyelvén, hatékonyan vannak megvalósítva. Ezen túl a VisualDSP++ támogatja az ún. beépített függvények használatát. Ezek a szabvány ANSI C nyelvben nem szereplő, azonban Blackfin processzor aritmetikai egységéhez szorosan kapcsolódó műveletek használatát egyszerűsítik. Ilyen például a fixpontos törtszámok (fract16 és fract32) szorzása, ahol fordító a C kódban szabványos függvényhívásnak megfelelő kódrészletet egyetlen, vagy mindössze néhány – a BF537 processzor architektúrájához jól illeszkedő – gépi utasításra cseréli le. C++ nyelv esetén a beépített függvények még hatékonyabb kihasználására van lehetőség.

4.3.3. Hibakeresés és tesztelés

A szoftverfejlesztési idő jelentős részét a megírt kódok tesztelése, a belekerült hibák felderítése és kijavítása teszi ki. Ezeket a feladatokat megfelelő eszközök nélkül szinte lehetetlen elfogadható minőségben, véges idő alatt elvégezni.

A VisualDSP++ az alap hibakereső szolgáltatásokon túl további hasznos lehetőségeket kínál. Így nem csak töréspont (breakpoint) elhelyezésére, léptetésre (step) és a változók figyelésére (variable watch) van mód, hanem a processzor megállított (halt) állapota mellett megvizsgálhatjuk az éppen futtatott kódrészlet gépi kódra fordított változatát (disassembly), valamint a processzor regisztereinek és bármely memóriaterületének tartalmát. A memória összefüggő részein tárolt értékek ábraként megjelenítve (pl. jelalakokat kirajzolva) is nyomon követhetők. Ezeken túl hasznos eszköz a beépített *statisztikai kiértékelő (statistical profiler)*, ami gyors áttekintést ad arról, hogy a program futása során az egyes kódrészleteket mennyi ideig, milyen arányban

használta a processzor.

Az alkalmazás fejlesztése és hibakeresése során figyelmen kívül hagyható, hogy fordítás után milyen környezetben futtatjuk, teszteljük a programot. A VisualDSP++ környezetben kiválaszthatjuk, hogy milyen céleszközön kívánjuk futtatni a megírt alkalmazást. A céleszköz lehet szimulátor, USB-n keresztül az EZ-KIT fejlesztőkártya, vagy JTAG emulátor használatával saját hardver.

4.4. VisualDSP++ Kernel operációs rendszer

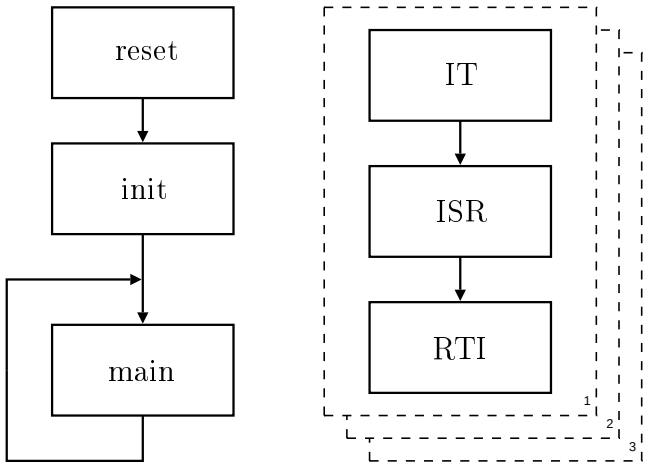
A beágyazott alkalmazások legfontosabb jellemzője, hogy szoros kapcsolatban vannak a külvilággal. A DSP ilyenkor a környezet bizonyos paramétereit méri, az ezen végzett számítások eredményét pedig tárolja, továbbküldi valamilyen kommunikációs interfészen, vagy beavatkozójel előállítására használja fel. A környezettel közvetlen kapcsolatban álló mérő és beavatkozó moduloknak általában szigorú időzítési követelményeknek kell teljesíteniük, felépítésük ugyanakkor egyszerű. Ezzel szemben a kommunikácós feladatokra a kevésbé kötött időzítés és a lényegesen bonyolultabb, esetenként több szoftverrégetből álló felépítés jellemző. Az elkészítendő alkalmazás ütemezési struktúrájának ehhez jól kell illeszkednie.

Sok esetben nagyon egyszerű struktúra alkalmazható: *reset* után az *inicializáiós* programszakasz kerül futtatásra, majd a *főprogram*, ami egy tétlen végtelen ciklus. Az alkalmazás hasznos kódrészleteire *megszakítások* segítségével kerül át a vezérlés, majd azok lefutása után automatikusan visszakerül a főprogramot jelentő üres hurokba, ld. 4.1. ábra.

Bonyolultabb beágyazott rendszerek tervezése során olyan további igények merülhetnek fel, mint a processzálás paramétereinek on-line állítása, a DSP erőforrásainak jobb kihasználása érdekében több komplex feladat egy processzoron történő párhuzamos elvégzése, vagy a hatékonyabb fejlesztés érdekében a kódújrahasznosítás és moduláris fejlesztés lehetőségének jobb kiaknázása. Ilyenkor célszerű egy teljesen más megközelítést eszközölni, az egész alkalmazást egy valós idejű operációs rendszer köré építeni. Jelenleg több beágyazott operációs rendszernek is van BF537 processzoron futó változata. Kézenfekvő választás ezek közül az Analog Devices saját fejlesztésű operációs rendszere, a VisualDSP++ Kernel (VDK) [6].

4.4.1. A VDK szolgáltatásai

A VDK szorosan integrálva van a VisualDSP++ fejleszői környezetbe, ami számos ponton gyorsítja az alkalmazásfejlesztést. A hardver inicializálá-



4.1. ábra. Egyszerű vezérlési szerkezet folyamatábrája

sát végző forráskódok automatikus generálása, a sablonfájlok alkalmazása és az eszközvezérlők egységesített progamozási interfészeinek használata jelentősen csökkenti a fejlesztési időt, az implementálás részleteinek elfedésével. A VDK programkódokat – mint a VisualDSP++ alkalmazásokat általában – írhatjuk C, C++ és assembly nyelven, biztosítva a kód olvashatóságát, karbantarthatóságát és optimalizálhatóságát.

Szálak (threads)

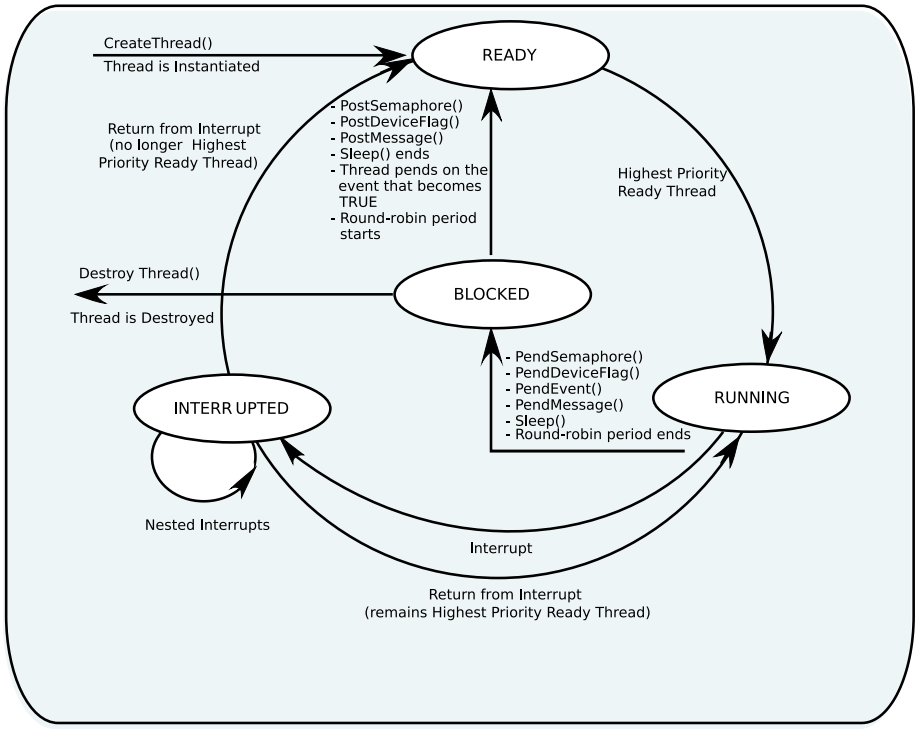
Az operációs rendszerek alapszolgáltatása, hogy szálak (vagy folyamatok) használatával az algoritmusok és a hozzá tartozó adatok egységes, de szálanként független kezelését biztosítja. Ezzel egy komplex rendszer olyan funkcionális blokkokra bontható, ahol az egyes folyamatok az összetett rendszernek csak egy jól meghatározott részfeladatáért felelősek. Az egyes szálak így külön-külön fejleszthetők és tesztelhetők, ami robusztusabb, jobban skálázható rendszer megépítését teszi lehetővé. A szálak a külvilág (további szálak és a hardver) felé jól meghatározott interfészen keresztül kommunikálnak, ezzel is segítve a kód-újrahasznosítást. Konkurrens folyamatok esetében ez a VDK *szignáljait*, míg hardver esetén a VDK *hardver absztrakciós rétegét* (hardware abstraction layer) jelenti.

Ütemezés (scheduling)

A VDK egy preemptív multitaszkos operációs rendszer. Ennek megfelelően a szálak elindításuk után csak a számukra definiált feladat végrehaj-

tásáért felelősek, az egyes szálak közötti kontextusváltás és versenyhelyzet kezelése előlük el van fedve, ez az operációs rendszer feladata.

Egy szál életciklusát a 4.2. ábra mutatja be. Belépésekor a szál *futásra kész (ready)* állapotban indul és felkerül a futászra kész folyamatok listájára (ready queue). Az ütemező innen választja ki azt, hogy melyik nyeri el a processzor használati jogát. Az éppen *futó (running)* folyamat addig birtokolja a processzort, amíg önszántából le nem mond róla (sleep), erőforrás hiányában várakozó, *blokkolt (blocked)* helyzetbe nem kerül, vagy egy megszakítás érvényrejutása következtében *megszakított (interrupted)* állapotra jut. A szál az erőforrás felszabadulását vagy a megszakítsi rutinok lefutását követően ismét futásra kész állapotba kerül. Ez alól a szál kilépése jelent kivételt, amikor is a blokkolt állapot után a szál végleg befejezi futását.

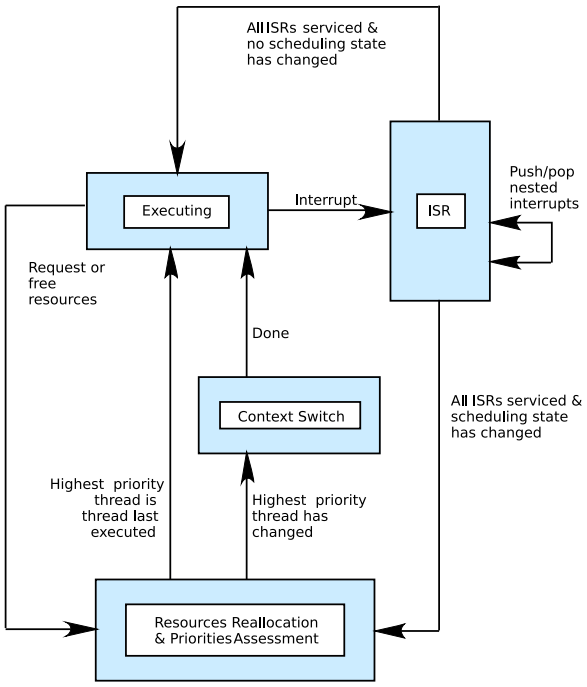


4.2. ábra. A szálak állapotdiagramja

A VDK indulásakor egyetlen szál van jelen, ami elvégzi az erőforrások inicializálását, majd belépési pontot biztosít az esetleges további szálak számára. Innentől kezdve a VDK *ütemező algoritmusa* szerint futnak az egyes kódrészletek, lásd 4.3. ábra. A VDK minden folyamathoz egy prioritásértéket rendel. Az arbitráció során a futásra kész folyamatok között található legna-

gyobb prioritású szál kerül futtatásra. Arbitrációra akkor van szükség, ha az éppen futó folyamat (executing) blokkolt állapotba kerül, vagy az őt félbeszakító ISR lefutása alatt erőforrások szabadultak fel. Amennyiben az aktuálisan futó folyamatnál nincsen magasabb priorituású, folytathatja munkáját. Ellenkező esetben az ütemező kontextusváltást (context switch) kezdeményez a legnagyobb prioritású folyamat javára.

Egyes algoritmusoknak vannak kódrészletei, melyek zavartalan lefutását garantálni kell. Az ilyen szakaszokat a más szálra váltás, vagy a hardveres megszakítás érvényre jutása darabolhatja fel. A VDK esetében a kontextusváltás ellen az ütemező időszakos kikapcsolása jelent megoldást (unscheduled region), ami biztosítja, hogy más szálhoz nem kerül át a futás joga. A hardveres megszakítások érvényre jutása hasonló módon tiltható le (critical region). Az így védett régió futása alatt beérkező megszakítási kérelmek csak a szakasz lefutása után kerülnek feldolgozásra.



4.3. ábra. A VDK állapotdiagramja

Szignálok (signals)

A szálak közötti kommunikáció (inter process communication) és szinkronizáció lebonyolítására a VDK négyféle szinkronizációs objektumot biztosít.

A *szemafor (semaphore)* az egyik legáltalánosabb szinkronizációs eszköz. Segítségével szabályozható a közös erőforrásokhoz való hozzáférés, szinkronizálhatóak az egyes szálak és jelezhetők bizonyos rendszeresemények. A VDK támogatja a periodikus szemaforok használatát is, ami a szálak periodikus futtatásához használható.

Az *üzenet (message)* elsősorban a szálak közötti kommunikáció eszköze. Használatával tetszőleges típusú információ vihető át az egyik folyamat adatterületéről a másikéra.

Az *eseményjelző bitek (event bits)* a rendszer állapotának jelzésére szolgálnak. A különböző rendszerállapotokra a szálakban különböző viselkedési stratégia definiálható.

Az *eszköz flag (device flag)* a szemaforhoz hasonló szignál, amit kifejezetten azért vezettek be, hogy az eszközvezérlők (device drivers) speciális viselkedését egyszerűen lehessen implementálni.

Megszakítások (interrupts)

Az megszakítási rutinokkal szemben elvárás, hogy mennél gyorsabban reagáljanak az őket generáló külső eseményre és mennél kevesebb ideig rabolják a processzor idejét. A megszakítást kiszolgáló kód (Interrupt Service Routine, ISR) ennek megfelelően rendkívül rövid és – bár a VDK támogatja a C és C++ nyelvű implementálást – általában alacsony szintű, hardverközeli nyelven kerül megírásra. A megszakítási rutinok tervezésénél irányelv, hogy a kód csak a leglényegesebb feladatokat végezze el (pl. buffer kiolvasása és eltárolása) és a megszakítási esemény bekövetkeztét szignálok segítségével jelezze az egyes szálak felé. Az bonyolultabb számításokat ezután a megfelelő – magasabb szintű nyelven megírt – szálban végezzük el. Ezzel a technikával jelentősen csökkenthető a kontextusváltások száma.

Hardverkezelés (I/O interface)

A perifériákkal közvetlen kapcsolatban álló kódrészletek teremtik meg a kapcsolatot a külvilág és az alkalmazásunk lényegi algorimusa között. Ezek pontos, hatékony megírása és a feldolgozást végző algoritmusoktól való átgondolt elválasztása kritikus az egész alkalmazásra nézve. A VDK ezt a *hardver absztrakciós réteg (hardware abstraction layer)* bevezetésével támogatja. Ennek alapötlete, hogy a környezettel kommunikáló hardver és az alkalmazás

szálai közé egy elválasztó szoftverréteget definiál. Ezt az réteget az eszközvezérlők (device drivers) valósítják meg. Az eszközvezérlők egységes interfészt mutatnak a szálak felé, így ezek hardverfüggő implementálációs részletei elfedhetők az alkalmazás folyamatai elől. Ezzel nem csak a főprogram és az eszközvezérlők fejlesztését tudjuk szétválasztani, hanem az alkalmazást is könnyen portolhatóvá tettük.

4.4.2. lwIP stack

Az 1970-es évek elejétől a lokális számítógép-hálózatok (Local Area Network, LAN) egyre nagyobb számban és terjedelemben kezdtek megjelenni a világban. Ezeket a helyi hálózatokat a sokféleség és az inkompatibilitás jellemezte, egészen az Ethernet V1.0 szabvány megjelenéséig, ami végül olyan domináns szerepet harcolt ki magának a LAN-ok területén, hogy a néhány év múlva IEEE 802.3 szabványt is az alapján készítették el. Ez a szabvány, és továbbfejlesztései a 80-as évek végétől napjainkig szinte egyeduralkodóvá vált a helyi számítógépes-hálózatok közt és mára a LAN kifejezés szinte egyet jelent az Ethernet / IEEE 802.3 szabvánnyal. Napjainkban sorra merülnek fel az igények, hogy kisebb eszközöket (pl. szenzorokat) is csatlakoztassunk *közvetlenül* ezekhez a kiforrott és már meglévő hálózatokhoz – és rajtuk keresztül akár az Internethez –, azonban ezen eszközök korlátozott erőforrásai ezt korábban nem tették lehetővé. Ma már kaphatók olyan beágyazott eszközök és szenzorok, melyek olcsók, fizikai méretük kicsi, és rendelkeznek olyan erőforrásokkal, hogy egy leegyszerűsített protokoll implementációt alkalmazva képesek legyenek az IEE 802.3 szabványban rögzített módon kommunikálni.

Az lwIP a TCP/IP protokoll stack [4] egy leegyszerűsített implementációja. Tervezésénél fő célkitűzés volt a memória felhasználás és a kódméret olyan mértékű minimalizálása, hogy alkalmas legyen a csekély erőforrással rendelkező eszközökben történő felhasználásra. Az lwIP más TCP/IP implementációkhoz hasonlóan rétegezett protokoll modell alapján készült. Az egyes protokollrétegeket különálló modulok valósítják meg. A rétegek szolgáltatásai az egyes modulok függvényeinek hívásával érhetőek el. Ezek a modulok az adott protokollnak csak a legfontosabb funkcióit tartalmazzák és bár minden protokollréteghez egyértelműen tartozik egy modul, ezek egyes függvényei áthágják a rétegek határait a sebesség és memóriahasználat optimalizásálásának érdekében. Így például ha a TCP modulnak szüksége van a cél és forrás IP címekre, akkor ezeket nem az IP modul függvényhívásai révén szerzi meg, hanem az IP fejléc struktúrájának ismeretében közvetlenül.

Az lwIP alkalmazásprogramozói felület (Application Programming Interface, API) a hálózati bufferek (network buffer) és kapcsolatok (network connection) kezelésére két adattípust és számos ezeken operáló függvényt de-

finiál. A *netbuf* típus a hálózati kommunikáció során használt bufferek egy absztrakt struktúrája. Az LWIP optimalizálásainak fő forrása, hogy az LWIP a buffereket (netbufs) nagyon erőforrástakarékosan kezeli azáltal, hogy az adatmásolások és mozgatások számát minimálisra csökkenti. A *netconn* a hálózati kapcsolatok absztrakt típusa, amihez UDP és TCP kapcsolatokat rendelhetünk. A hálózati kapcsolatok létrehozása és kezelése a *netconn* típus kezeléséhez tartozó API függvényekkel lehetséges. Az LWIP hálózatkezelő API használata nagyon hasonló a Berkley Socket Distribution (BSD) socket API kezelésénél megszokottakkal, implementációja azonban több helyen egyszerűsítéseket tartalmaz.

A VDK egyik előnye a gyors alkalmazásfejlesztés. Ez nagyon gyakran a már korábban megírt programrészletek újrafelhasználásával érhető el. Amennyiben rendelkezésre áll egy olyan könyvtár, ami ugyanazzal az interfésszel rendelkezik, mint a BSD socket API, akkor a már meglévő hálózatkezelő algoritmusok újrafelhasználása azok egyszerű átmásolását jelenti. A VisualDSP++ Kernel a hálózati kapcsolatok kezelését az LWIP TCP/IP stack segítségével valósítja meg. A VDK az LWIP fölé egy további réteget definiál, mely réteg a programozó felé ez BSD socket API-val kompatibilis interfészként látszik. A hálózati alkalmazás fejlesztése így – bizonyos korlátok szem előtt tartásától eltekintve – teljesen hardverfüggetlenül történhet. A VisualDSP++ környezet további szolgáltatása, hogy a TCP/IP stack paramétereit egy plugin segítségével grafikus felületen is beállíthatjuk, ami alapján a hálózatkezeléshez tartozó inicializációs forráskódot automatikusan elkészíti.

5. fejezet

Rendszerterv

5.1. Hardver rendszerterv

A CDFT algoritmus hardveren történő megvalósításához kétféle rendszerterv készült. A tervek középpontjában mindkét esetben az Analog Devices EZ-KIT fejlesztőkártyája és a rajta lévő ADSP-BF537 jelfeldolgozó processzor állt.

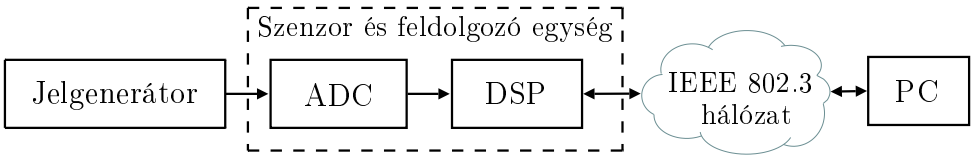
5.1.1. Alap rendszer

Az alap rendszerterv kizárólag a CDFT algoritmus DSP-n történő kipróbálására készült, ennek blokkvázlatát ld. 5.1. ábrán. A dolgozat készítése során ezt implementáltam, a további fejezetekben ezzel foglalkozom.

A jelgenerátoból érkező jel a fejlesztőkártya AD1854 típusú 48 kHz frekvenciával mintavételező ADC konverterére kerül, majd onnan a DSP-re. Az ábrán látható a szaggatott vonal a fejlesztőkártyát jelöli. A processzor első lépésként a nem-egyenletes mintavételezést szimulálja azáltal, hogy a minták csak egy részét tartja meg. A megtartott mintákat a rendszer időbélyeggel látja el az aktuális rendszeridő alapján. A DSP az ilyen időbélyeg-minta párokat dolgozza fel a korábban ismertetett CDFT algoritmus szerint. A feldolgozás végeredményt a kártyával egy Ethernet hálózaton található PC-ről lehet lekérdezni. Ugyancsak a személyi számítógépről lehet a DSP-n futó algoritmus paramétereit állítani.

5.1.2. Bővített rendszer

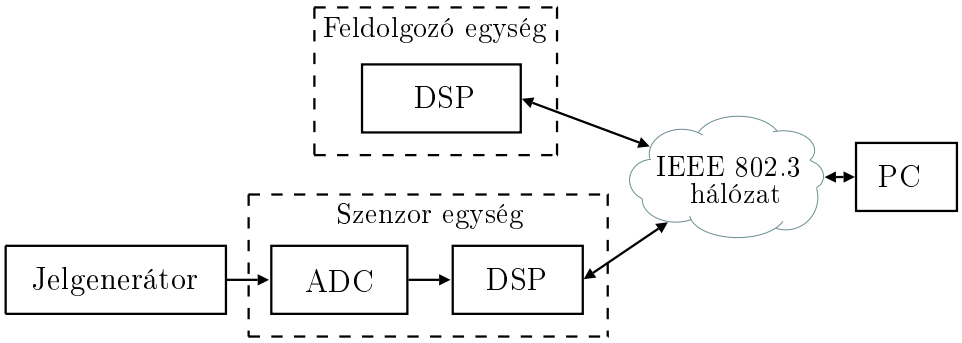
A második, bővített rendszertervet a CDFT algoritmus elosztott jelfeldolgozó hálózaton történő tesztelésének céljából hoztam létre. Ez mutatja



5.1. ábra. A CDFT algoritmus DSP-n futását demonstráló rendszer felépítése

be a továbbfejlesztési lehetőségeket, azonban teljes implementációjára jelen dolgozat keretein belül nem került sor.

A jelfeldolgozó hálózat fő építőelemei a hálózatra csatlakoztatható szenzorok, az adatfeldolgozást végző egység és maga a hálózat. Tervem a szenzort és a feldolgozó egységet már fizikailag elkülönítve tartalmazza. Ezt szemléltei az 5.2. ábra. A hálózatra csatlakoztatható érzékelőt és a feldolgozó algoritmust futtató hardvert külön-külön EZ-KIT fejlesztőkártya valósítja meg. Ehhez a hálózathoz szintén csatlakozik egy személyi számítógép, ami a korábbi lekérdezésen és paraméterállításon túl monitorozási funkciót is betölt. A szenzorok, a spektrumbecslést végző hardver és a PC közötti kommunikáció ebben az elrendezésben egy dedikált, közös IEEE 802.3 hálózaton keresztül zajlik, ez az hálózat azonban alkalmas vezeték nélküli hálózatok, vagy Internetes kapcsolatokat szimulációjára is.

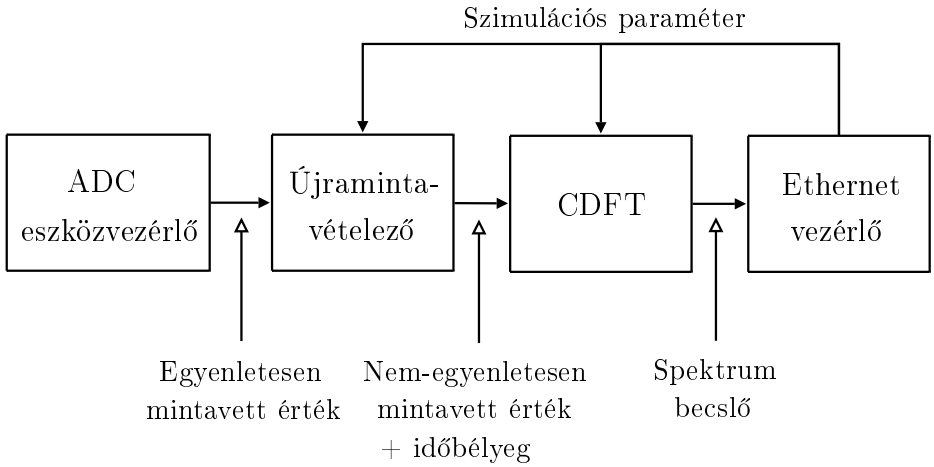


5.2. ábra. A CDFT algoritmus jelfeldolgozó hálózatban futását demonstráló rendszer felépítése

5.2. Szoftver rendszerterv

A rendszer szoftverének elkészítéséhez a korábban ismertetett VDK valós idejű operációs rendszert választottam. Döntésemet elsősorban az indokolta,

hogy a bonyolult Ethernet kommunikációt lebonyolító LWIP stack rendszerbe integrálásának ez volt a egyetlen módja. Továbbá a feldolgozás egyes fázisait így jól szeparálva, külön-külön funckionális blokkokra tudtam osztani. A funkcionális blokkok, modulok közötti közötti hatásdiagram az 5.3. ábrán látható. Az ábrán található blokkok a VDK operációs rendszerben egy-egy szálnak felelnek meg. Ezek a VDK szignáljait használva kommunikálnak és szinkronizálnak egymáshoz.



5.3. ábra. Szoftver rendszerterv – hatásdiagram

Az *ADC eszközvezérlő* a jel mintavételezését és mért értékek eszközfüggetlen formátumú továbbítását végzi az *újramintavételező* szál felé. Az *újramintavételező* szál ezután a beállított szimulációs paraméterek alapján a minták ritkításával szimulálja a nem-egyenletes mintavételezést. A megtartott mintákat időbélyeggel látja el, majd továbbítja a spektrumbecslést végző CDFT szál felé. A CDFT szál a saját paramétereit figyelembe véve elvégzi a számításokat, majd az eredményt továbbküldi az *Ethernet vezérlő* szálnak. Végül az *Ethernet vezérlő* a spektrumbecslés eredményét telnet üzenetek formájában elküldi a hálózaton.

A következőkben ezen négy szál interfésze és funkcionális leírása kerül részletes ismertetésre.

5.2.1. ADC eszközvezérlő

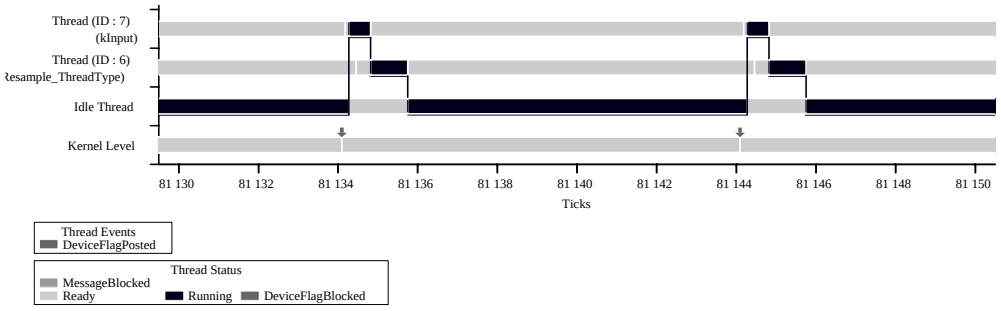
Az *ADC eszközvezérlő (ADC device driver)* a feldolgozási folyamat első fázisát képezi, ahogy az 5.3. ábrán is látható. Feladata a mért értékek lekérése az ADC-től és ennek továbbítása az újramintavételező szál felé. Az ADC

Irány	Típus	Leírás
Bemenet	ADC regiszter	48 kHz frekvenciával egyenletesen mintavett értékek
Kimenet	VDK üzenet	48 kHz frekvenciával egyenletesen mintavett értékek

5.1. táblázat. Az ADC eszközvezérlő interfésze

eszközverzérlő a VDK hardver absztrakciós rétegének filozófiáját követve készült el, tehát működése többnyire gépi kódban van megadva, a külvilág felé pedig magasszintű, hardverfüggetlen felületet mutat.

Az eszközvezérlő működése az 5.4. ábrán látható futási diagram segítségével követhető nyomon. A vízszintes tengelyen az rendszeridő, a függőlegesen pedig az egyes szálak állapota és a hozzájuk tartozó események találhatók. A konstans 48 kHz mintavételi frekvenciával működő ADC minden mérés alkalmával megszakítást generál, amit a futási diagramon a Kernel level sávján található nyíl előtti szakadás jelez. A megszakítás a VDK-ban az ADC eszközvezérlőhöz van rendelve. A megszakításhoz tartozó ISR fejlesztőkártya specifikus, az ADC regisztereit olvassa be, majd jelzi ezt egy eszköz flag bebillentésével. A flag bebillentését a Kernel level sávon látható nyíl jelöli a futási diagramon. Ennek hatására a vezérlés átkerül az eszközvezérlő további kódrészleteire, amit az 5.4. ábrán az eszközvezérlőhöz tartozó kInput szálra váltás mutat. A szál a beolvasott értékeket 48 mintából álló bufferbe gyűjti, majd szabványos VDK üzenetként a többi szál rendelkezésére bocsátja. A további szálak elől így a hardverfüggő részek teljesen el vannak rejtve.



5.4. ábra. Az eszközvezérlő futási diagramja

5.2.2. Újramintavételező

Az *újramintavételező (Resample)* szál a nem-egyenletes mintavételezés szimulációját végzi az ADC eszközvezérlőtől kapott adatok ritkításával. A szimuláció kimenetét jelentő új mintavételi pontok egymástól vett távolsága egyenletes eloszlás szerint alakul. Ennek paraméterei a pontok távolságának átlagos értéke és szórása, amelyek a szál paraméterei.

Irány	Típus	Leírás
Bemenet	VDK üzenet	48 kHz frekvenciával egyenletesen mintavett értékek
Bemenet	VDK üzenet	szimulációs paraméterek
Kimenet	VDK üzenet	szimulált nem-egyenletes időbeni eloszlású minták időbélyeggel ellátva

5.2. táblázat. Az újramintavételező szál interfésze

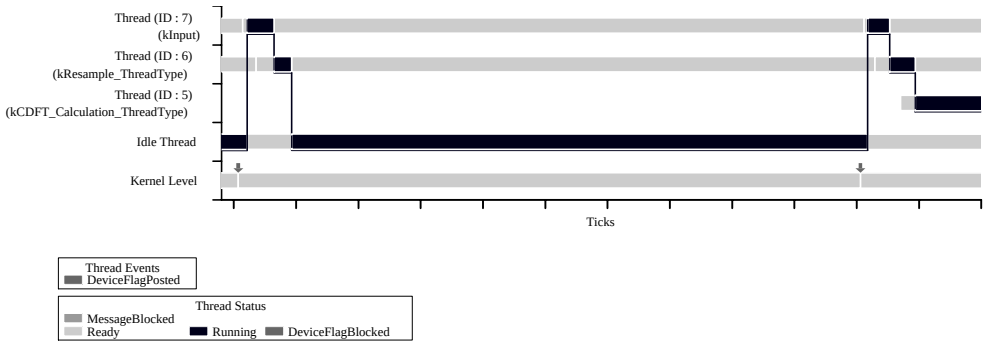
Az ADC eszközvezérlőtől kapott 48 mintából álló bufferek közül véletlenszerűen kerül kiválasztásra, hogy melyik mért érték kerül továbbításra. A véletlen kiválasztás paraméterei az Ethernet vezérlő szálból VDK üzenetekkel állíthatók. A továbbküldésre szánt üzenetek ebben a fázisban kapnak időbélyeget, amit a szál a rendszeridőből nyer ki.

Az újramintavételező működését az 5.5. ábra mutatja. A képen a mintaritkítás két lehetséges esete látható. A baloldalon található eseménysorozat az az eset, amikor a minta nem kerül továbbküldésre. Itt a Resample szál megkapja a mintát VDK üzenet formájában az eszközvezérőhöz tartozó kInput száltól, azonban ezt nem küldi tovább, a vezérlés így nem kerül át a CDFT kiértékelő folyamatra. Ezzel szemben a jobboldalon a kiválasztott üzenetet továbbítja kInput a CDFT szál felé, így az megkezdi a feldolgozást.

5.2.3. CDFT kiértékelő

A *CDFT kiértékelő (CDFT calculation)* szál minden egyes időbélyegminta páros beérkezése után kiszámolja a spektrum becslőjét a CDFT algoritmus segítségével, majd a paraméterként megadott periodicitással továbbküldi azt az Ethernet vezérlő szálnak.

A szál és az egész alkalmazás magját a spektrumbecslést végző CDFT algoritmus képzi. Az algoritmushoz tartozó függvények és változók egységbezárásának céljából egy külön osztályt (class CDFT) hoztam létre. A szál indulásakor ezt példányosítja, és az így kapott tagobjektum metódusát hívva számítja a spektrumot. A metódus futási ideje lényegesen hosszabb, mint a



5.5. ábra. Az újramintavételező futási diagramja

Irány	Típus	Leírás
Bemenet	VDK üzenet	szimulált nem-egyenletes eloszlású minták időbélyeggel ellátva
Bemenet	VDK üzenet	a spektrumbecslési eredmény továbbküldésének gyakorisága
Kimenet	VDK üzenet	a spektrumbecslés eredményét tartalmazó frekvencia- és komplex amplitudóvektor címe

5.3. táblázat. A CDFT kiértékelő szál interfésze

többi szál által végzett feladatnak, ezért a szál viszonylag kis prioritással, mintegy háttérfolyamatként végzi a számításokat. A CDFT osztály és a hozzá tartozó CDFT algoritmus számítását végző metódus részletes ismertetése az 5.3 alfejezetben található.

5.2.4. Ethernet vezérlő

Irány	Típus	Leírás
Bemenet	telnet üzenet	szimulált nem-egyenletes eloszlású minták időbélyeggel ellátva
Bemenet	VDK üzenet	gyakorisága
Kimenet	VDK üzenet	a belső kéréseknek megfelelő szöveges formátumú telnet üzenet

5.4. táblázat. Az Ethernet vezérlő szál interfésze

Az *Ethernet vezérlő (Ethernet control)* szál a hálózati kommunikációt bonyolítja le. Belépési pontja a többi folyamathoz hasonlóan az inicializációs szálban van, azonban a többitől eltérően ez nem statikusan hívódik meg, hanem a telnet protokollnál használt (well-known) 23-as TCP portra érkező kapcsolódási kérelem beérkezésekor automatikusan. Ez a kapcsolatot a kártyával egy hálózaton található személyi számítógépről kezdeményezhető, egy egyszerű terminál kliens segítségével. A szál indulása után rövid üdvözlőszöveget küld a kiépített kapcsolaton keresztül. Ezután lehetőség van a mérés és feldolgozás paramétereinek egyszerű paranccsal történő beállítására, majd a feldolgozó algoritmus indítására. Indulása után az egyes modulok (más szálak) periodikusan küldött üzeneteit, így többek között a spektrumbecslőt továbbítja a szál a PC-re a telnet kapcsolaton keresztül.

5.3. A CDFT algoritmus implementálása

A CDFT algoritmust számító függvény és a hozzá tartozó állapotváltozók jól elkülöníthetők az alkalmazás többi részétől, ezért célszerű ezt egységesen, de az alkalmazás többi elemétől elkülönítve kezeleni. A egységbezárást elősegítendő egy külön osztályt (`class CDFT`) hoztam létre az algoritmusnak. Az alkalmazásban ezt példányosítva, majd a megfelelő tagfüggvényt hívva lehet futtatni az algoritmust.

5.3.1. CDFT osztály definíciója

Az osztály tagjai a CDFT algoritmust számító függvény, a hozzá tartozó állapotváltozók és az inicializálást végző konstruktor. A tagváltozókat két csoportba lehet osztani. Az egyik csoportba a konstansok tartoznak, vagyis az olyan változók melyek a inicializáláskor értéket kapnak, és ezután ez a program futása során nem változik. Ezekre első sorban a futási idejű osztások kiiktatása céljából van szükség. Ilyenek tagok az $1/2\pi$ és az $1/k$ értéket tartalmazó shortfract[1] típusú változók, ahol $k = 1, ..., N-1$ és N jelöli a kiértékelendő frekvenciák száma. Használatukkal az algoritmus osztás műveletei reciprokszorzásra cserélhetők. A másik csoportot a folyamatosan frissülő tagváltozók alkotják. Ezek például a beérkezett mintákat, időbélyegeket, vagy számított komplex spektrumot tároló tömbök. Az osztály teljes definícióját a következő kódrészlet mutatja be.

[1] a shortfract a fract16 C++ csomagoló osztálya (wrapper class)

```
class CDFT {
   private:
      shortfract x[NS];
      float t[NS];
      shortfract t_fract[NS];
      shortfract t_arg[NS][NF];
      float t_sum;
      int idx;
      shortfract twopi_rec;
      shortfract sin[NS][NF];
      shortfract cos[NS][NF];
      shortfract omega_rec[NF];
   public:
      float f[NF];
      complex_shortfract y[NF];

      CDFT();
      ~CDFT();
      int cdft_zoh( float ts, fract16 x );
};
```

5.3.2. A cdft_zoh tagfüggvény definíciója

A CDFT algoritmust számító tagfüggvény neve `cdft_zoh`. A függvény bemenete egy float típusú időbélyeg és egy hozzá tartozó fract16 típusú mért érték, kimenete pedig két tömb. Egyik tömb tartalmazza azon frekvenciaértékeket, melyeken a spektrumbecslő kiértékelésre került, másik az ezen frekvenciákon kiszámított komplex spektrumbecslőt. Az interfészről 5.5. táblázat ad áttekintést.

Irány	Típus	Leírás
Bemenet	float	legfrissebb mintához tartozó időbélyeg (másodpercben)
Bemenet	fract16	legfrissebb mért érték
Kimenet	float	frekvenciaértékeket tartalmazó tömb
Kimenet	complex_shortfract	spektrumbecslőt tartalmazó tömb

5.5. táblázat. A CDFT algoritmust számító függvény interfésze

Az CDFT futási szakaszait az 1. algoritmus ismerteti, ld. következő oldal. Ebből az *inicializációs* résznek az osztály konstruktora, a *huroknak (loop)* pedig a `cdft_zoh` tagfüggvény felel meg. A hurokban lévő utasítások minden

inicializálás
loop
x_{t_N} minta eltárolása
for $k = 1$ to $N - 1$ **do**
$X(f_k) = -\frac{1}{j2\pi f_k} \sum_{i=0}^{N-1} x_{t_i}(e^{-j2\pi f_k t_{i+1}} - e^{-j2\pi f_k t_i})$
end for
$X(0)$ számítása
end loop

Algoritmus 1: A CDFT pszeudokódja

új kiértékelendő minta beérkezésekor végrehajtásra kerülnek, ami egyet jelent a `cdft_zoh` függvény futtatásával.

A függvény működésének és az alkalmazott optimalizálások részletes tárgyalásához tekintsük annak forrását, ld. alábbi kódrészlet.

```
int CDFT::cdft_zoh( float ts_, fract16 x_ ) {

   // Store new element
   t[idx] = ts_;
   x[idx] = x_;

   // Prescale timestamps
   t_fract[idx] = (ts_*HZ/NF)-floor(ts_*HZ/NF);

   // Calculate fract16 arguments
   int i;
   t_arg[idx][0] = 0;

   for ( i = 1 ; i < NF ; i++ ) {
      // Unsaturated sum of the fract16's
      t_arg[idx][i] = (fract16)(((int)t_arg[idx][i-1]
                    + (int)t_fract[idx]) & 0x7FFF);
   }

   // Calculate harmonics
   int omega;
   for ( omega = 1 ; omega < NF ; omega++ ) {

      // Clear y
      y[omega].re = 0.0;
      y[omega].im = 0.0;
```

```
      // Calculate 1/(2*pi)*exp(j*omega*t(i))
      sin[idx][omega] = (shortfract)sin2pi_fr16(t_arg[idx][omega])
                      * twopi_rec;
      cos[idx][omega] = (shortfract)cos2pi_fr16(t_arg[idx][omega])
                      * twopi_rec;

      // Sum of multiplied exponential differencies
      for ( i = idx+1 ; i < idx+NS ; i++ ) {

         y[omega].re += (fract16)(
                          (sin[(i+1)%NS][omega]-sin[i%NS][omega])
                        * x[i%NS]*omega_rec[omega] );
         y[omega].im += (fract16)(
                         -(cos[(i+1)%NS][omega]-cos[i%NS][omega])
                        * x[i%NS]*omega_rec[omega] );
      }
   }

   // Calculate DC
   y[0].re = 0.0;
   y[0].im = 0.0;

   t_sum = t[idx] - t[(idx+1)%NS];
   if ( t_sum > 0) {
      float dc = 0.0;
      for (i = idx+1 ; i < idx+NS ; i++ )
         dc += (float)x[i%NS]*(t[(i+1)%NS]-t[i%NS]);
      dc = dc/t_sum;
      y[0].re = (shortfract)dc;
   }
   else {
      y[0].re = 0;
   }

   idx++;
   idx %= NS;
   return 0;
}
```

A függvény első lépésben eltárolja a beérkezett x_{t_N} értéket és a hozzá tartozó t_N időbélyeget a megfelelő tömbökbe (bufferekbe). A bufferekben a tárolás cirkulárisan van szervezve, azaz egy mutató jelzi, hogy éppen melyik

pozícióban van az aktuálisan beérkezett minta. A mutató értéke minden ciklusban növekszik, míg a buffer végére nem ér. A buffer végének elérésekor a mutató ismét az első elemre ugrik. A bufferen végzett műveletek mindig a mutatóhoz relatívan kerülnek végrehajtásra. Ezzel a technikával rengeteg adatmozgatás megtakarítható.

A következő lépés az

$$X(f_k) = -\frac{1}{j2\pi f_k} \sum_{i=0}^{N-1} x_{t_i}(e^{-j2\pi f_k t_{i+1}} - e^{-j2\pi f_k t_i}) \qquad k = 1, 2, ..., N-1 \quad (5.1)$$

kiértékelése minden f_k frekvenciára, ami

$$f_k = k\Delta f \qquad k = 1, 2, ..., N-1 \quad (5.2)$$

esetén

$$X(f_k) = -\frac{1}{j2\pi k\Delta f} \sum_{i=0}^{N-1} x_{t_i}(e^{-j2\pi k\Delta f t_{i+1}} - e^{-j2\pi k\Delta f t_i}) \qquad k = 1, 2, ..., N-1 \quad (5.3)$$

alakban írható. Az f_k frekvenciák ilyen megválasztása teljesen kézenfekvő.

A számítások során szeretnénk a DSP architektúrájához jól illeszkedő fract16 típust használni. A fract16 típus csak a $[-1, 1)$ tartományt képes árbázolni, és azt is korlátozott felbontással, ezért használatánál körültekintően kell eljárni. A további lépések ennek fényében kerültek megtervezésére.

Az (5.3) kifejezés kiértkelésének első fázisa az exponenciális függvény argumentumának meghatározása. A $2\pi k\Delta f t_i$ kifejezés számítását érdemes a t_i időminták Δf értékkel való szorzásával kezdeni. Ezt a függvény a minta és az időbélyeg eltárolása után azonnal megteszi.

Vegyük észre, hogy az exponenciális függvény 2π szerint periodikus. Ennek megfelelően ekvivalens kifejezést kapunk, ha az argumentumot csökkentjük 2π egész számú többszörösével. Az időminták értékét így mindig a $[0, 2\pi)$ tartományban tarthatjuk. A függvény az időmintát ennek megfelelően csonkolja. Az implementáció során a $[0, 2\pi)$ intervallum helyett $[0, 1)$ tartománnyal találkozunk. Ennek oka, hogy a szinusz és koszinusz függvények kiértékelése a gépi ábrázoláshoz szorosan kötődő `sin2pi_fr16` függvénnyel kerülnek kiszámításra. Ez a függvény $[0, 1)$ intervallumból származó fract16 bemeneti értéket vár, és ezt $[0, 2\pi)$ intervallumból származó értékként értelmezi.

A `cdft_zoh` függvény következő lépésben egy egyszerű `for` ciklusba szervezett összeadással kiszámítja a korábban meghatározott argumentum $k > 1$ egész számú többszöröseit, ahol szintén kihasználja a `sin2pi_fr16` függvény

gépi ábrázolásban $[0, 1)$ szerinti periodicitását. Ezeket a számítási részeredmények tárolásra kerülnek, mintánként csak egyszer kell meghatározni. Az argumentumok előállítása után már minden frekvenciára meghatározható az $e^{-j2\pi f t_i}$ kifejezés. Ez az

$$e^{jx} = \cos(x) + j\sin(x) \tag{5.4}$$

összefüggés szerint, azaz Euler formula segítségével történik. Ez mintánként minden frekvencián csak egyszer kerül kiértékelésre, ezeket a részeredményeket szintén tárolja a CDFT objektum. Miután az exponenciális tagokat megkaptuk, a szummában fennmaradt kivonást, szorzást[2] egy `for` cikluson belül elvégezzük, az eredményt pedig folyamatosan akkumuláljuk. A cikluson belül jól megfigyelhető a cirkuláris bufferkezelés. Ezzel a spektrum periodikus komponenseit meghatároztuk.

A függvény utolsó lépésként a $X(0)$ (DC) komponenst határozza meg. A DC szint elkülönített meghatározására azért van szüksége, mert hatékony számításánál nem elegendő a fract16 számábrázolási tartománya. A számítás menete kézenfekvő, a jel értékét kiintegráljuk az egész mintavett tartományra, a ZOH modell figyelembevételével:

$$X(0) = \frac{1}{t_N - t_0} \int_{t_0}^{t_N} x(t)\mathrm{d}t \simeq \frac{1}{t_N - t_0} \sum_{i=0}^{N-1} x_{t_i}(t_{i+1} - t_i) \tag{5.5}$$

5.3.3. Továbbfejlesztési lehetőségek

Az itt bemutatot rendszertervről elmondható, hogy a célkitűzéseimnek megfelelően moduláris felépítésű. Az így megvalósítot alkalmazás egyes szoftverkomponensei egymástól függetlenül, könnyen cserélhetők és módosíthatók. Így a rendszer kommunikációs interfésze, jelfeldolgozási feladat rugalmasan változtatható.

A CDFT módszert implementáló algoritmus azonban csak minimális optimalizálást tartalmaz. Ez elsősorban azt jelenti, hogy lényeges részeket fract16 típussal valósítottam meg, az ehhez kapcsolódó számábrázolási problémákat megoldottam. A következőkben néhány kézenfekvő optimalizálási lehetőség kerül felsorolásra:

[2] A törtszámok szorzása a $*$ operátorral történik. Ez azért helyes, mert *shortfract* objektumokat használunk, ahol a $*$ operátor felül van definiálva. Külön érdekesség, hogy ez a fordító beépített függvényének hívásával történik, így a művelet minimális számú gépi utasításra fordul le.

- *Szinusztábla használata.* Az algoritmus futásához érdemes a trigonometrikus függvények számításánál alkalmazott könyvtári függvényhívásokat szinusztáblázatra cserélni. Ezt a hardver erőforrások lehetővé teszik.

- *További részeredmények tárolása.* A kód hatékonyabbá tehető, ha a (2.15) egyenletet nem ennyire direkt módon programozzuk le, azaz – a jelenlegi implementációban letárolt exponenciális tagokon túl – további részeredményeket is megőrzünk a későbbi számításokhoz.

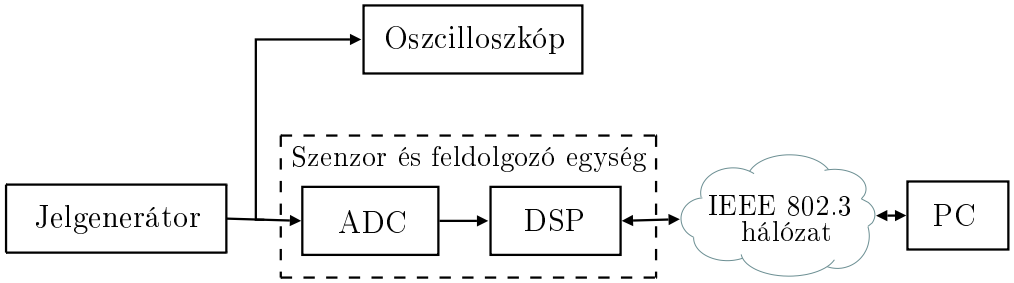
- *Szorzások számának csökkentése.* Az algoritmus jelenlegi implementációja (2.15) kifejezésnek megfelelően tartalmaz egy $j2\pi$ tényezőt a nevezőben. Ez a gyakorlati alkalmazásokban elhagyható.

6. fejezet

Eredmények

A fejezetben a korábban bemutatott demonstrációs rendszerrel készített mérések és eredményeik kerülnek ismertetésre, majd az így kapott eredményeket összehasonlítom a Matlab szimulációval kapható eredményekkel.

A demonstrációs rendszeren végzett mérések elvégzéséhez egy függvénygenerátort, egy oszcilloszkópot, egy személyi számítógépet és egy EZ-KIT fejlesztőkártyát használtam. Az függvénygenerátor kimenetét az oszcilloszkópra és a fejlesztőkártya ADC konverterének bemenetére vezettem, ld. 6.1. ábra. Az oszcilloszkóp a mérés során csak ellenőrző funkciót töltött be. A mérőkártyához a számítógép egyszerű crosslink kábellel csatlakoztattam, ez alkotta az egyszerű Ethernet hálózatot. A mérési eredményeket az itt felépített telnet kapcsolaton keresztül nyertem ki és rögzítettem. Az XML struktúrában rögzített adatokat ezután egy Matlab parser scripttel dolgoztam fel.

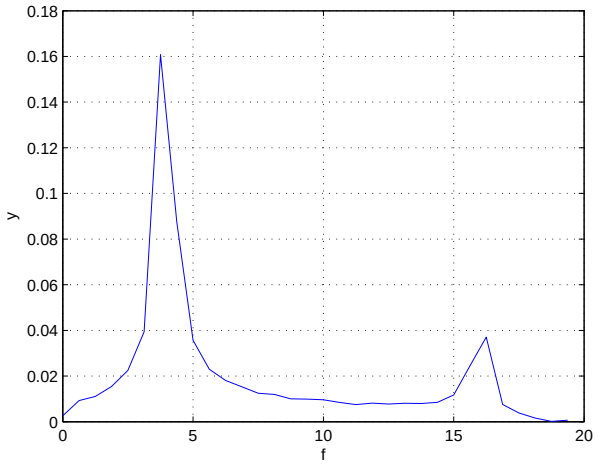


6.1. ábra. A CDFT algoritmus DSP-n futását vizsgáló mérési összeállítás

A mérések során az algoritmus futását kétféle gerjesztőjel mellett regisztráltam. A gerjesztőjelek frekvenciája mindkét esetben a 10 Hz nagyságrendbe estett. A frekvenciaértéket azért választottam ilyen alacsonyra, mert a jelenlegi nem optimalizált implementáció nem enged meg 20-30 Hz mintavé-

teli frekvenciánál gyorsabb feldolgozást. Ennek megfelelően az demonstrációs rendszer gyakorlati alkalmazásokra még nem alkalmas, de a működés demonstrációs célokra megfelel.

Az első esetben sima 4 Hz szinuszjelet alkalmaztam. A mérést egyenletes és szimulált jitteres mintavételezéssel végeztem el. A rendszer mintavételi frekvenciája egyenletes mintavételezésnél 20 Hz, jitteres esetben $20 \pm 5\,\mathrm{Hz}$. A bufferelt minták és a számított spektrumkomponensek száma egyaránt 32. Ilyen beállítások mellett a 6.2. ábrán látható eredményet kaptam.



6.2. ábra. $f_s = 20$ Hz mintavételi frekvenciával egyenletesen mintavételezett 4 Hz szinuszjel CDFT spektrumbecslője DSP-vel számítva

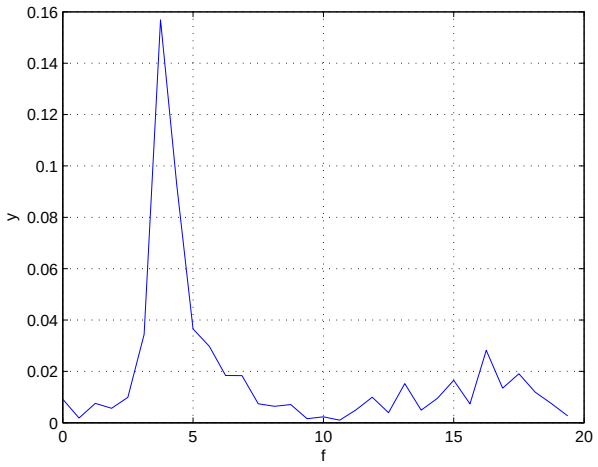
A mérés eredményét jitteres, nem-egyenletes mintavételezés esetén 6.3. ábra mutatja.

A bemutatott 6.2 és 6.3. ábrák a DSP által számolt spektrum pillanatképei. Ennek időbeni alakulását a 6.4 és 6.5. ábrán lévő spektrogram szemlélteti.

Az így kapott mérési eredményeket érdemes összevetni a szimulációk szolgáltatta értékekkel, ld. 3.2. ábra. A két eredmény teljesen megegyezik, ami igazolja az DSP-s implementáció sikerességét.

Ezután olyan mérést állítottam össze, ami a frekvenciatartomány f_s mintavételi frekvencia feletti részének analízisét teszi lehetővé. A mintavételi frekvenciát lecsökkentettem 10 Hz-re, a spektrumot pedig 40 Hz-ig számítottam, 64 pontban. A mérés eredményét a 6.6 és 6.7. ábrák tartalmazzák az egyenletes és a nem-egyenletes mintavétel esetére.

Mindkét ábrán jól látszik a sinc burkoló hatása. Látható, hogy a nem-egyenletes esetben a 16 Hz környékén látható komponens a nem-egyenletes

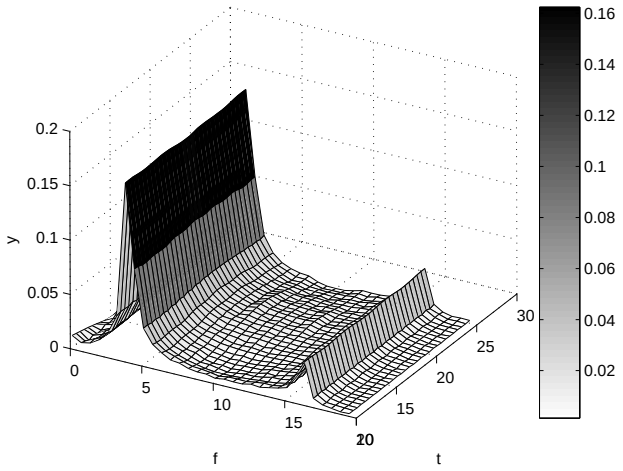


6.3. ábra. 4 Hz nem-egyenletesen mintavételezett szinuszjel CDFT spektrumbecslője DSP-vel számítva

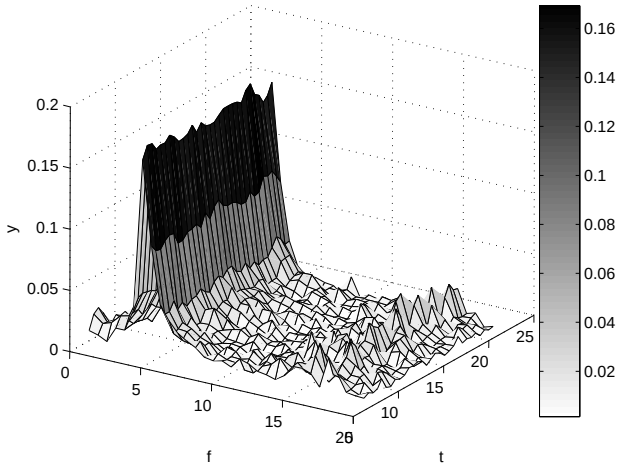
mintavételi frekvencia miatt kevésbé markáns, zajszerűbb.

A 6.8 és 6.9. ábrákon a függvénygenerátor *sweep* üzemmódja mellett rögzített mintákból számított spektrumok látahtók. A szinuszjel frekvenciáját így 15 másodperc alatt 2.5Hz – 5Hz között egyenletes sebességgel változtattam. Ennek egyenletes és nem-egyenletes esetben készített spektrogramját mutatja a 6.8, 6.9 és 6.10. ábra.

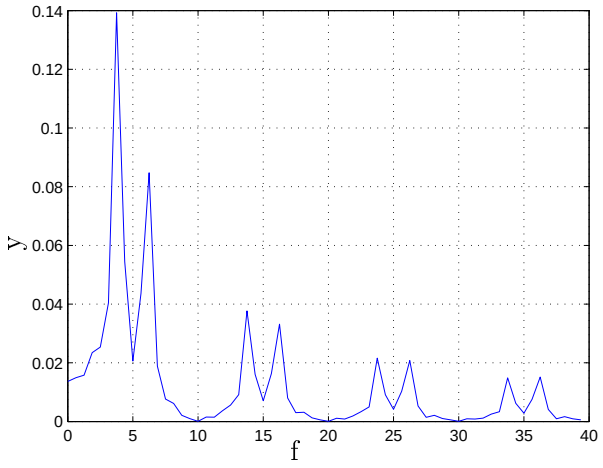
A demonstrációs rendszerrel készített eredmények tehát a szimulációk során kapott eredményeknek megfelelnek, ezzel bizonyítják az implementáció sikerességét. Ugyanakkor rámutatnak az algoritmus megvalósítási problémáira és a korábban feltárt optimalizálási lehetőségek megvalósításának szükségességére.



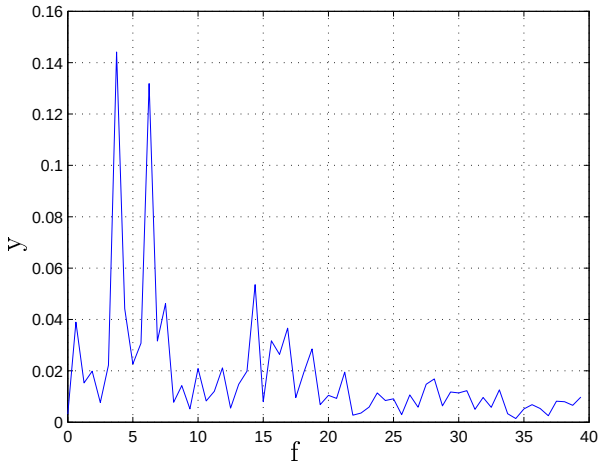
6.4. ábra. 4 Hz egyenletesen mintavételezett szinuszjel CDFT spektrogramja, DSP-vel számítva



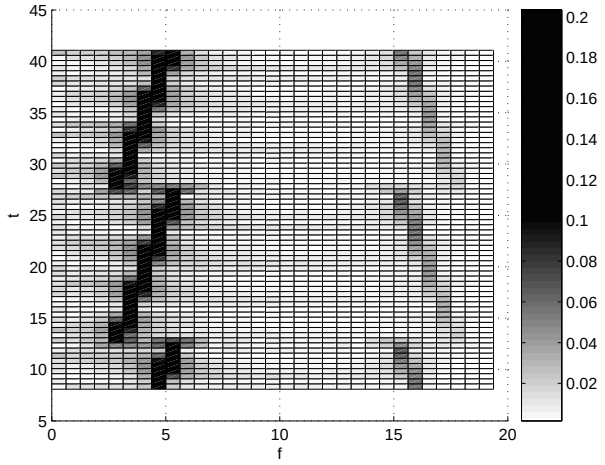
6.5. ábra. 4 Hz nem-egyenletesen mintavételezett szinuszjel CDFT spektrogramja, DSP-vel számítva



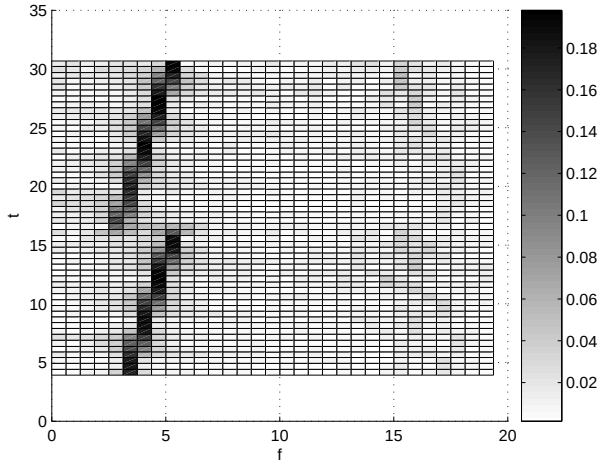
6.6. ábra. 4 Hz egyenletesen mintavételezett (f_s=10Hz) szinuszjel CDFT spektruma $4f_s$-ig, DSP-vel számítva



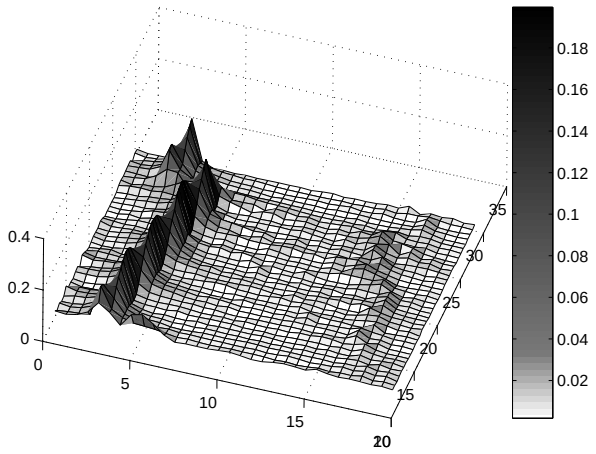
6.7. ábra. 4 Hz nem-egyenletesen mintavételezett szinuszjel CDFT spektruma $4f_s$-ig, DSP-vel számítva



6.8. ábra. A 2.5-5Hz között egyenletesen változó, egyenletesen mintavételezett ($f_s = 10$ Hz) szinuszjel CDFT spektrogramja $4f_s$-ig, DSP-vel számítva



6.9. ábra. A 2.5-5Hz között egyenletesen változó, nem-egyenletesen mintavételezett ($f_s \simeq 10$ Hz) szinuszjel CDFT spektrogramja $4f_s$-ig, DSP-vel számítva



6.10. ábra. A 2.5-5Hz között egyenletesen változó, nem-egyenletesen mintavételezett ($f_s \simeq 10$ Hz) szinuszjel CDFT spektrogramja $4f_s$-ig, DSP-vel számítva

7. fejezet

Összefoglalás

A dolgozat bemutatta a jelfeldolgozásban általában és a szenzorhálózatban gyakran felmerülő nem-egyenletes mintavételezési problémák típusait, valamint ismertette az egyes problématípusokra alkalmazható módszereket.

Matlab szimulációk segítségével összehasonlította a CDFT algoritmust a gyakorlatban leginkább alkalmazott újramintavételezés és FFT módszerrel. Ezután bemutatta a módszer egy implementációját az ADSP BF537 fixpontos jelfeldolgozó processzoron, valamint a demonstrációhoz szükséges egy lehetséges mérési elrendezést.

A megvalósítás eredményei egyelőre csak korlátozott mértékben használhatók a gyakorlatban, tekintettel a jelenlegi implementációval elérhető maximális mintafeldolgozási sebességre. A dolgozat rámutatott a jelfeldolgozó algoritmus szoftver implementációjának szűk keresztmetszeteire, gyengeségeire és javaslatot tett ezek kijavítására. Létrejött egy alkalmazás, melynek moduláris kialakítása lehetővé teszi az egyes komponensek egymástól független továbbfejlesztését vagy cseréjét.

Az így létrehozott rendszer alapja lehet a tanszéken folyó további munkáknak. Több DSP kártya hálózatba kapcsolásával ez a próbarendszer egy megfelelő platform elosztott jelfeldolgozási algoritmusok implementálására.

Irodalomjegyzék

[1] *IEEE 1451 Standards for smart transducer interface for sensors and actuators.* http://www.ieee1451.nist.gov.

[2] *IEEE 1588 Standard for a Percision Clock Synchronization Protocol for Network Measurement and Control Systems.* http://www.ieee1588.nist.gov.

[3] Lee A. Barford. *Fourier Transform for Timestamped Network Data.* U.S. Patent, Nr. 6735539, 2001.

[4] Adam Dunkels. *Design and Implementation of the lwIP TCP/IP Stack.* Swedish Institute of Computer Science, 2001.

[5] Analog Devices Inc. *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors.* Analog Devices Inc., 2005.

[6] Analog Devices Inc. *VisualDSP++ 4.0 Kernel (VDK) User's Guide.* Analog Devices Inc., 2005.

[7] Analog Devices Inc. *EZ-KIT Lite for Analog Devices ADSP-BF537 Blackfin Processor.* Analog Devices Inc., 2006.

[8] Analog Devices Inc. *ADSP-BF534/ADSP-BF536/ADSP-BF537 Blackfin Embedded Processor Data Sheet.* Analog Devices Inc., 2007.

[9] N. R. Lomb. *Astrophysics and Space Science.* vol. 39, pp. 447–462,. 1976.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipies in C.* pp. 575-584. 1992.